

Cloud Computing

Virtual Resources and Distributed Cloud Applications

Chapter 9: Observability



Distributed and Operating Systems
Electrical Engineering and Computer Science
Technische Universität Berlin

9. Monitoring und Observability

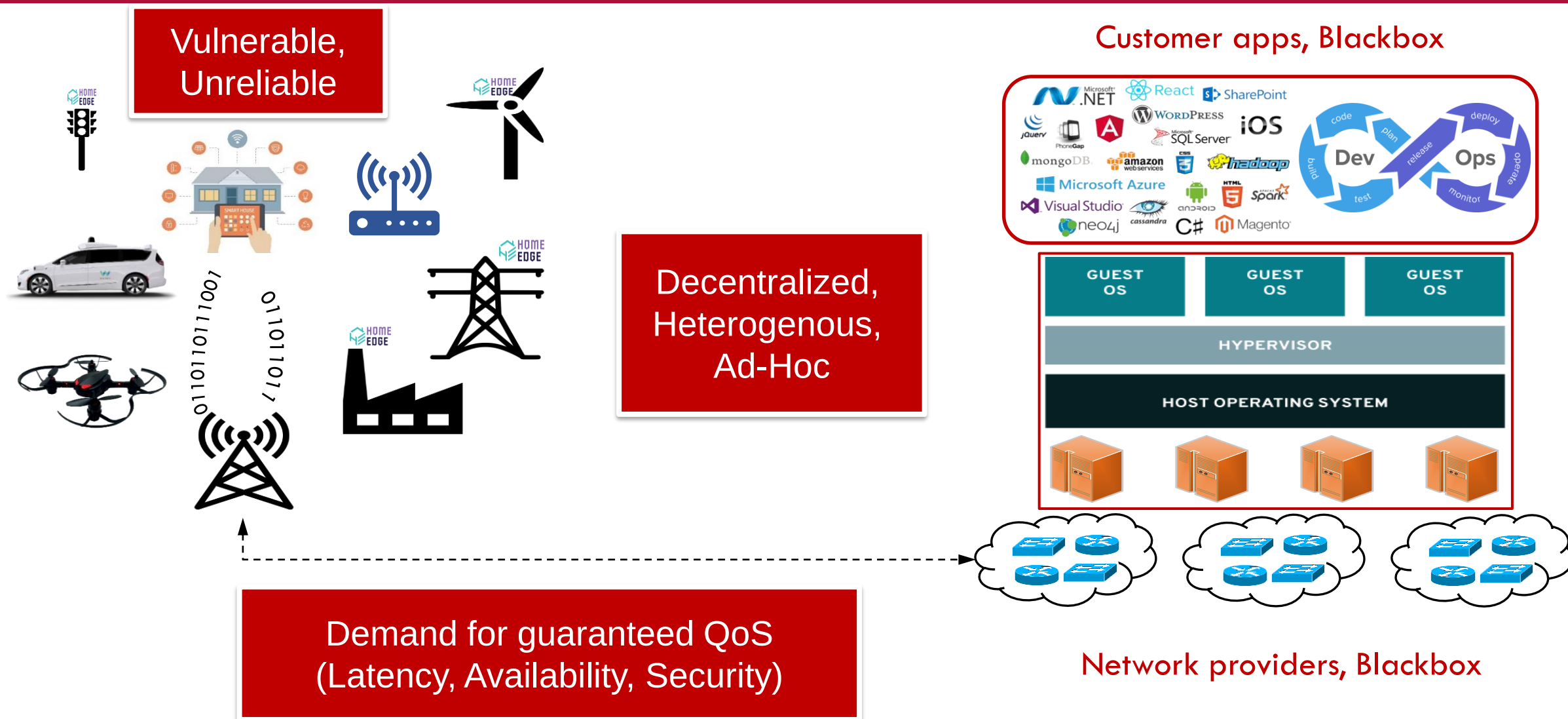
- Overview
 - 9.1 Definitions and data types
 - 9.2 Monitoring
 - 9.3 Observability
 - 9.4 AI-enabled Operations (AIOps)
 - 9.6 AIOps and log data

Cloud Operation is a Challenge

Big data and fast data are solved!
*How to keep the gigantic clusters up and running is the
main challenge!*

Prof. Michael Stonebraker, MIT Computer Science & Artificial Intelligence Lab,
Turing price winner 2015

Cognitive Cloud Continuum



State Of The Internet LIVE

Failures are inevitable

Downtime cost for medium-sized companies

\$400.000/h

US

\$45.000/h

Germany

OUTAGES PER HOUR
0 +200

This is a live broadcast of Internet statistics, as collected by Pingdom from over 700,000 users across the world. [See how you can use Pingdom to monitor your website.](#)

Microsoft Office 365 Outage Reported For Some Users

The company says it has been 'applying steps to mitigate the issue' of connection timeouts for some mailbox users.

By [Kyle Alspach](#)

January 24, 2019, 12:37 PM EST



@SlackHQ

Replying to [@sarbbottam](#)

Sorry for the trouble! We'r the moment, and hope to | normal as quickly as we ca

INTEGRATED TELECOMMUNICATIONS SERVICES MARCH 3, 2021 / 5:51 PM / UPDATED 19 DAYS AGO

Zoom down for more than a thousand users - Downdetector

By Reuters Staff

1 MIN READ



Slack System Status
Resources for real-time
Slack service.
[status.slack.com](#)

Microsoft Teams down: video call service back online

By Mike Moore · 7 days ago

Microsoft Teams saw major outages, but should be back online now

10:32 PM · Jun 28, 2019 · [Sparkcentral.com](#)

Google Cloud Outage Attributed To 'Infrastructure Components' Issues

'We are experiencing an issue with Google Cloud infrastructure components,' Google Cloud confirmed in a post on its [Google Cloud Status Dashboard](#).

By [Donna Goodison](#)

March 26, 2020, 12:55 PM EDT

Facebook struggles to deal with epic outage



By [Donle O'Sullivan](#) and [Heather Kelly](#), [CNN Business](#)

Updated 0654 GMT (1454 HKT) March 14, 2019

TECH



Slack asks investors to trust that outage costs were a 'one-time' issue

PUBLISHED THU, SEP 5 2019-4:31 PM EDT

[Jordan Novet](#)
[@JORDANNOVET](#)

SHARE [f](#) [t](#) [in](#) [e](#)

Oops!

We're very sorry, but we're having trouble doing what you just asked us to do. Please give us another chance--click the Back button on your browser and try your request again. Or start from the beginning on our [homepage](#).



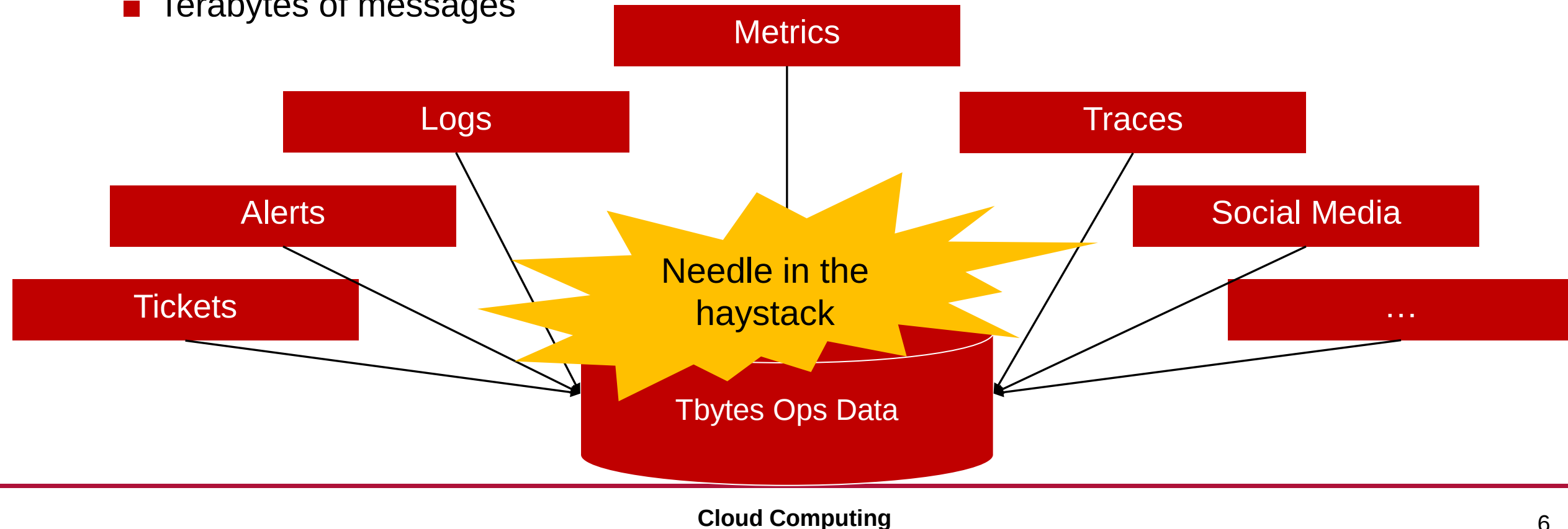
Cloud

5-minute outage costs Google \$545,000 in revenue

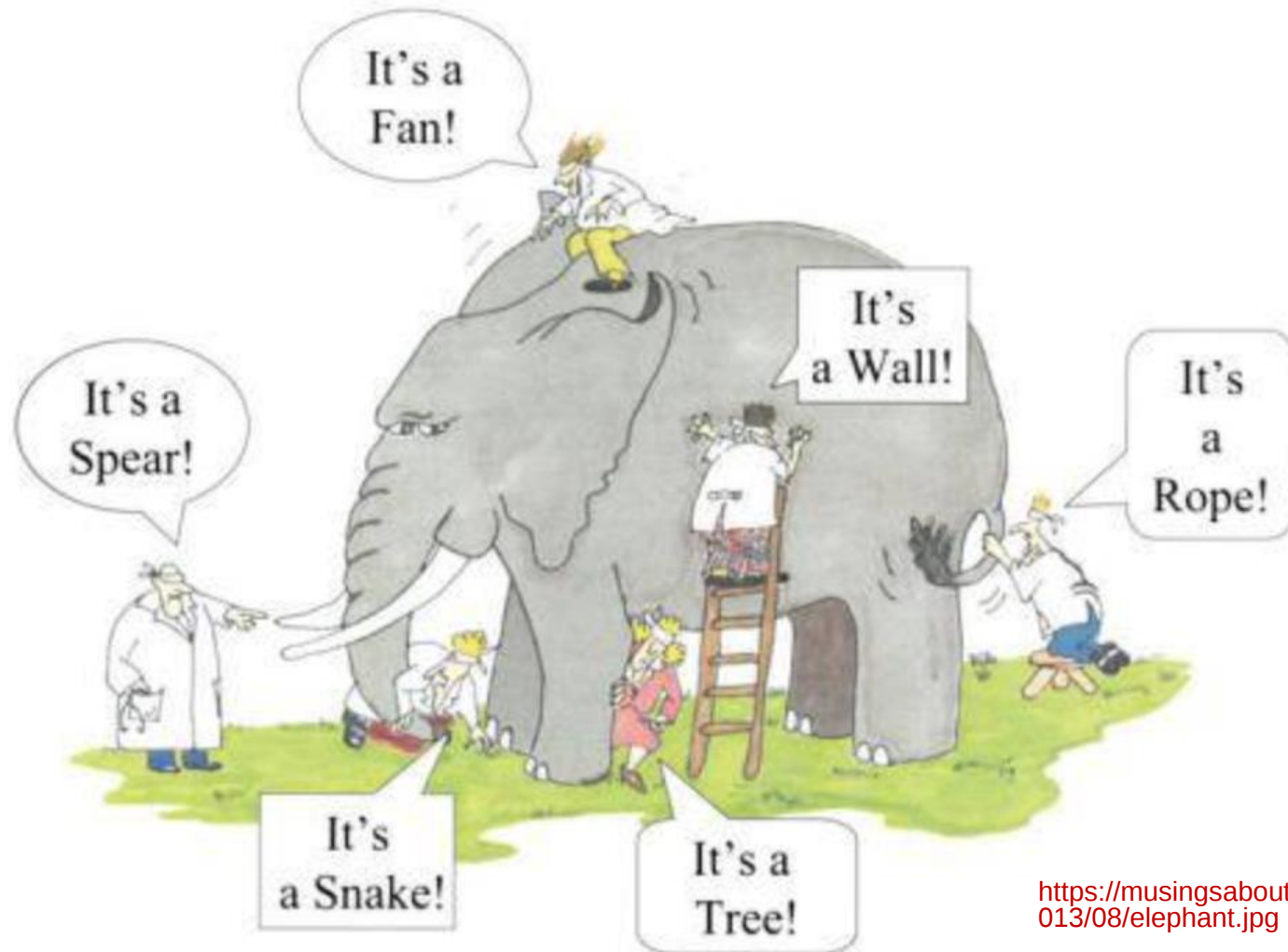
DYLAN TWENEY @DYLAN20 AUGUST 16, 2013 4:06 PM

The other side of agile development

- Rise of the DevOps leads to frequent changes, updates, upgrades, ...
 - Thousands software and config changes per day in a large company
 - Many sources of errors
 - Terabytes of messages



Multimodal Analysis counts



<https://musingsaboutsoftwaredevelopment.files.wordpress.com/2013/08/elephant.jpg>

9.1 Definitions: Monitoring and Observability

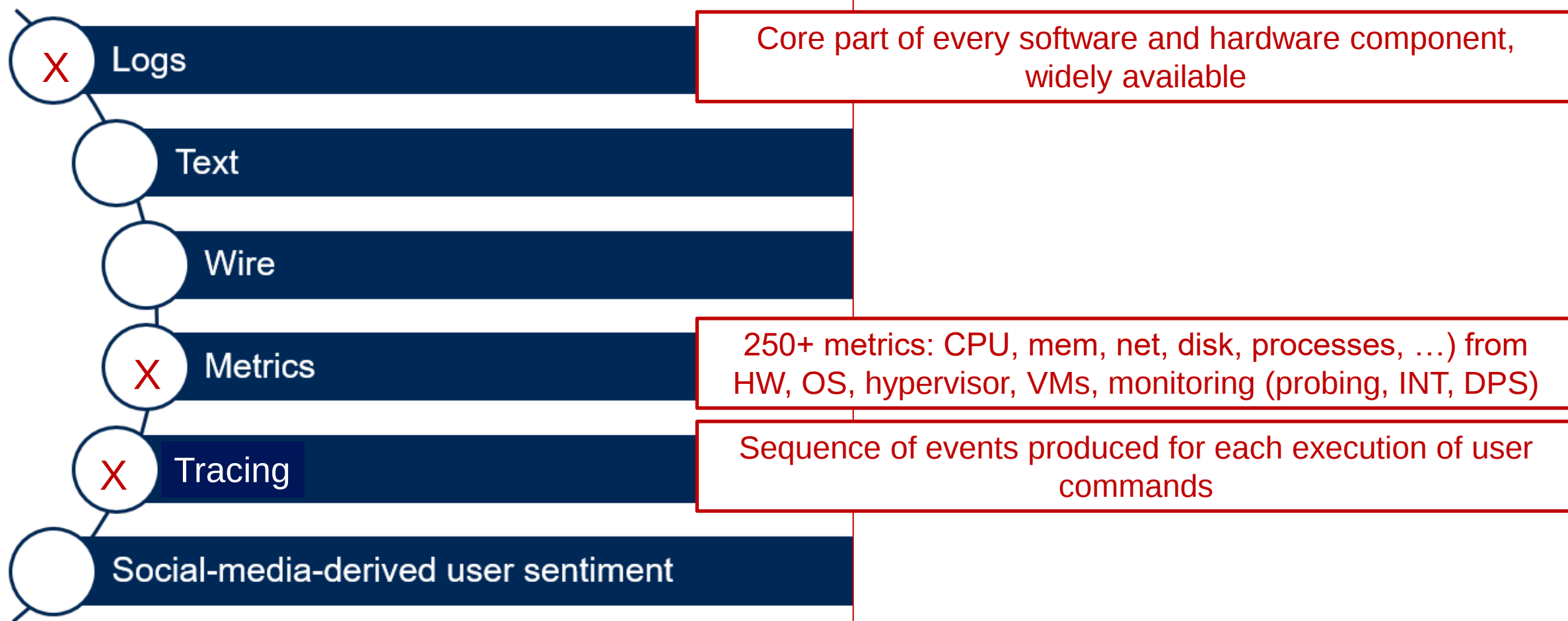
- Monitoring: Collects information and informs admins about detected issues based on pre-defined thresholds / rules → Looking for known or predictable failures
- Observability: Understanding what is happening inside a system by looking at the output data → helps to analyse this data to assess the system's health and find ways to fix problems in the IT infrastructure.
- Once initial observability is established, following actions are typically implemented
 - Developing actionable alerts
 - Creating useful dashboards,
 - Deploying AI-enabled solutions

Monitoring tells you whether a systems is working; observability lets you ask why it isn't working.

Definitions: Monitoring and Observability



Data is the Key



ID: 340492

Data types for observability

- Internal data
 - Metric data: numerical values about the health of the system and its components, for example CPU, memory or network utilization, CPU temperature and much more
 - Logs: textual notes created by developers to provide future users and administrators with advice on how to repair or understand the *_running_* system (while comments are suitable for the source code). Structured, unstructured, or plain text, these readable files can tell the results of any transaction involving an endpoint within icloud environment.
 - Traces: Automatically generated information that characterizes the path of the specific process execution, i.e. which method called which other methods with which parameters. Span data is a hallmark of tracing including unique identifiers, operation names, timestamps, logs, events, and indexes.
- External data
 - Tickets: Mails and phone calls by user to customer support
 - Tweets and all other types of user feedback on social media

9.2 Monitoring Distributed Systems

- Monitoring: Collecting, processing, aggregating, and displaying real-time quantitative data about a system, such as query counts and types, error counts and types, processing times, and server lifetimes.
 - White-box monitoring: based on metrics exposed by the internals of the system, including logs, interfaces like the JVM Profiling, or HTTP handler
 - Black-box monitoring: Testing externally visible behaviour
- Dashboard: Usually web-based application providing a summary view of core metrics using filters, selectors and exposing the metrics most important to admins
- Alert: notification intended to be read by a human → pushed to a system such as a bug or ticket queue, an email alias
- Root cause: origin of an error that latter propagates through the system causing additional symptoms

Monitoring: typical tasks

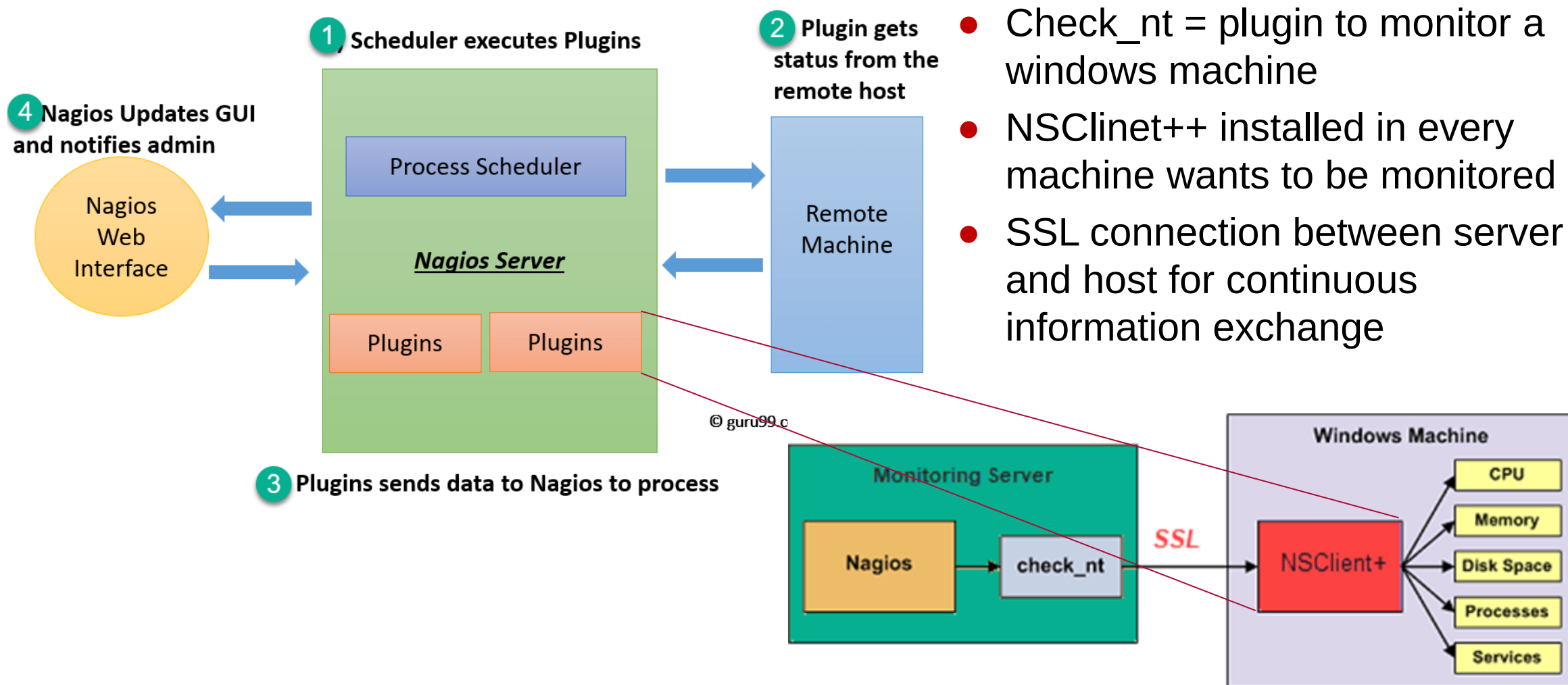
- Analysing long-term trends, e.g.
 - how big and how fast is the data store growing? How the number of active users evolving?
- Comparing over time or experiment groups, e.g.
 - are queries faster with technology X or technology Y? Is the site slower than it was last week?
- Alerting
 - Error happened, need fix immediately. Ideally, predictive maintenance is in place indicating an error in near future
- Conducting ad hoc retrospective analysis (i.e., debugging), e.g.
 - the latency just shot up; what else happened around the same time?
 - Monitoring is a significant engineering endeavour in and of itself. e.g. a Google SRE team with 10–12 members typically has one or two members whose primary assignment is to build and maintain monitoring systems for their service.

Symptoms Versus Causes

- Monitoring system addresses two questions: what is broken and why?
 - "what's broken" indicates the symptom
 - "why" indicates a (possibly intermediate) cause.

Symptom	Cause
I'm serving HTTP 500s or 404s	Database servers are refusing connections
My responses are slow	CPUs are overloaded, or an Ethernet cable is crimped under a rack, visible as partial packet loss
Users in Antarctica aren't receiving animated cat GIFs	CDN blacklisted some client IPs
Private content is world-readable	A new software push caused ACLs to be forgotten, allowed all requests

Monitoring Components on example Nagios



Types of Monitoring

1. System: performance of infrastructure parts at the physical layer
2. Dependency: creating graph of interacting system components based on network traffic / connections
3. Web performance: service response to a user-request at the client-side (page load speed, knowledge transmission errors, loading errors, etc.)
4. Integration and API: spot the provision and time period performance of third-party integrations
5. Application Performance: how well the apps behave within the current state of the IT surroundings
6. Real User: record user interactions and supply the historical performance of a service delivered to end-users over the network.
7. Security: uncommon activities may be tracked against outlined policies of knowledge access and authorization.

Examples for Types of Metrics

- Important: select non-correlated metrics, so each metric provides unique insight
 - Host-Based: most primitive metrics such as utilization of CPU, Memory, Disk space, Processes
 - Application: concerned with units of processing or work that depend on the host-level resources → highly application-specific, e.g. Error and success rates, Service failures and restarts, Performance and latency of responses, Resource usage
 - Network and Connectivity: services are accessible to other machines, e.g. connectivity, Error rates and packet loss, Latency, Bandwidth utilization
 - Server Pool: refers to horizontally scaled infrastructure as a higher-level extrapolation of application and server metrics, e.g. Pooled resource usage, Scaling adjustment indicators, Degraded instances
 - External Dependency: help to identify problems with providers that may affect the own operations, e.g. Service status and availability, Success and error rates, Run rate and operational costs, Resource exhaustion

Appropriate Resolution for Measurements

- Different aspects of a system measured with different levels of granularity
 - Observing CPU load over the time span of a minute won't reveal even quite long-lived spikes that drive high tail latencies.
 - Web service targeting no more than 9 hours aggregate downtime per year (99.9% annual uptime), probing for a 200 (success) status more than once or twice a minute is probably unnecessarily frequent
 - Checking hard drive fullness for a service targeting 99.9% availability more than once every 1–2 minutes is probably unnecessary.
- Frequent measurements may be very expensive to collect, store, and analyse.
- Reducing the costs by performing internal sampling on the server and including an external system to collect and aggregate that distribution over time, e.g. record the current CPU utilization each second and aggregate values every minute to observe brief CPU hotspots

Important Qualities of a Metrics

- Independent : external to other services with limited impact on the performance (typically, monitoring and AIOps limited to 2% of computing power)
- Reliable: gather all information continuously, as dropped metrics, service outages, and unreliable alerting lead to alert fatigue and admins ignore the alerts
- Historical data: monitoring is most useful when it has a rich history of data that can help establish trends, patterns, and consistencies over long timelines.
- Correlating factors from different sources; monitoring provides a holistic view of the infrastructure and must display related information from different characteristics → Admins glue information to understand potential interactions and overall infrastructure status (time synchronization a big issue here)
- Alerting: flexible enough to notify operators through multiple mediums and powerful enough to compose thoughtful, actionable notification triggers.

Some additional terminology

- Instrumentation: ability to track the behavior and performance of software by adding code and configuration to software to output data for a monitoring system
- Observer effect: impact of the monitoring system on the phenomena being observed → seek to avoid adding unnecessary overhead
- Over-monitoring: quantity of metrics and alerts configured is inversely related to their usefulness → causes stress on the infrastructure, make it difficult to find relevant data
- Alert fatigue: human response of desensitization that results from frequent, unreliable, or improperly prioritized alerts → operators ignore severe problems
- Threshold: boundary between acceptable and unacceptable values → alerts are configured to trigger when a value exceeds the threshold for a certain period of time, in order to avoid sending an alert for temporary spikes.

Four Golden Signals

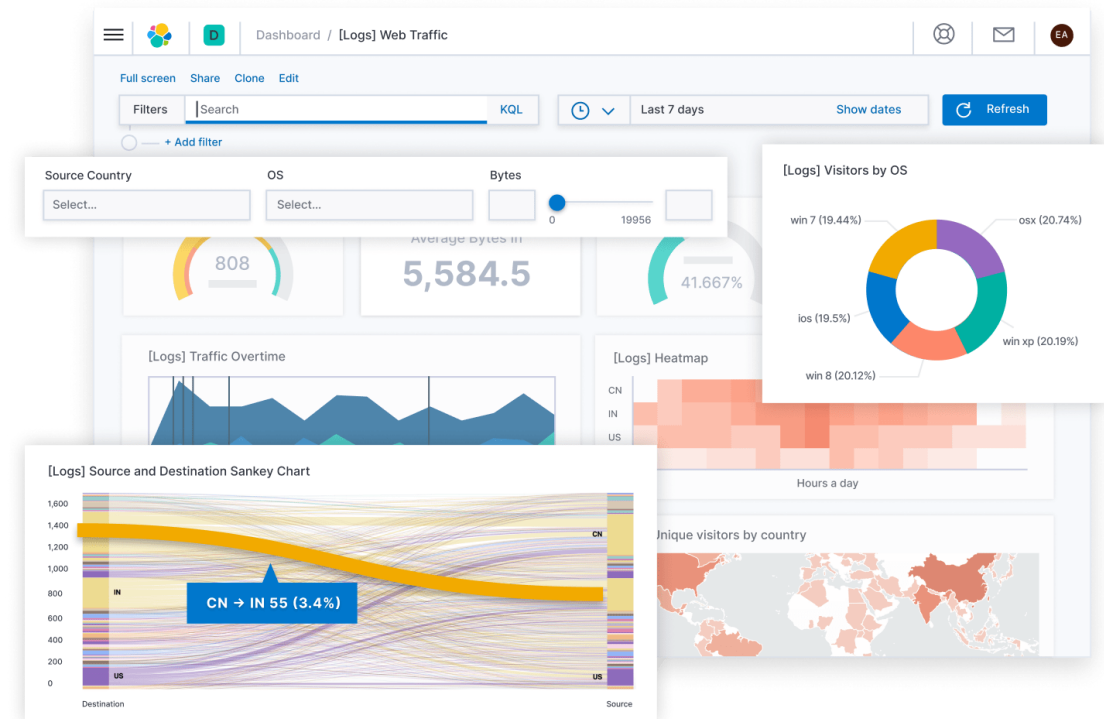
- Google pushes the theory that observing the following four most important monitoring signals is sufficient:
 1. Latency: time to service a request, traditionally distinguished between the latency of successful requests and the latency of failed requests
 2. Traffic: how much demand is being placed on the system, measured in a high-level system-specific metric (e.g. HTTP requests per second, network I/O rate, concurrent sessions, retrievals per second)
 3. Errors: rate of failed requests
 4. Saturation: How "full" is the service in terms of memory, CPU, network utilization. In complex systems, saturation can be supplemented with higher-level load measurement, e.g. can the service properly handle double the traffic or only 10% more traffic
- Typical monitoring approach: measure all four golden signals and alert an admin when one signal is problematic

As Simple as Possible

- Basic technologies
 - Alerts on different latency thresholds, at different percentiles, on all kinds of different metrics
 - Extra code to detect and expose possible causes
 - Associated dashboards for each of these possible causes
- Monitoring can become so complex that it's fragile, complicated to change, and a maintenance burden → simplicity and robustness are important
- Some guidelines:
 - Rules to catch real incidents should be as simple, predictable, and reliable as possible.
 - Data collection, aggregation, and alerting configuration that is rarely exercised should be up for removal.
 - Signals that are collected, but not exposed in any prebaked dashboard nor used by any alert, are candidates for removal.

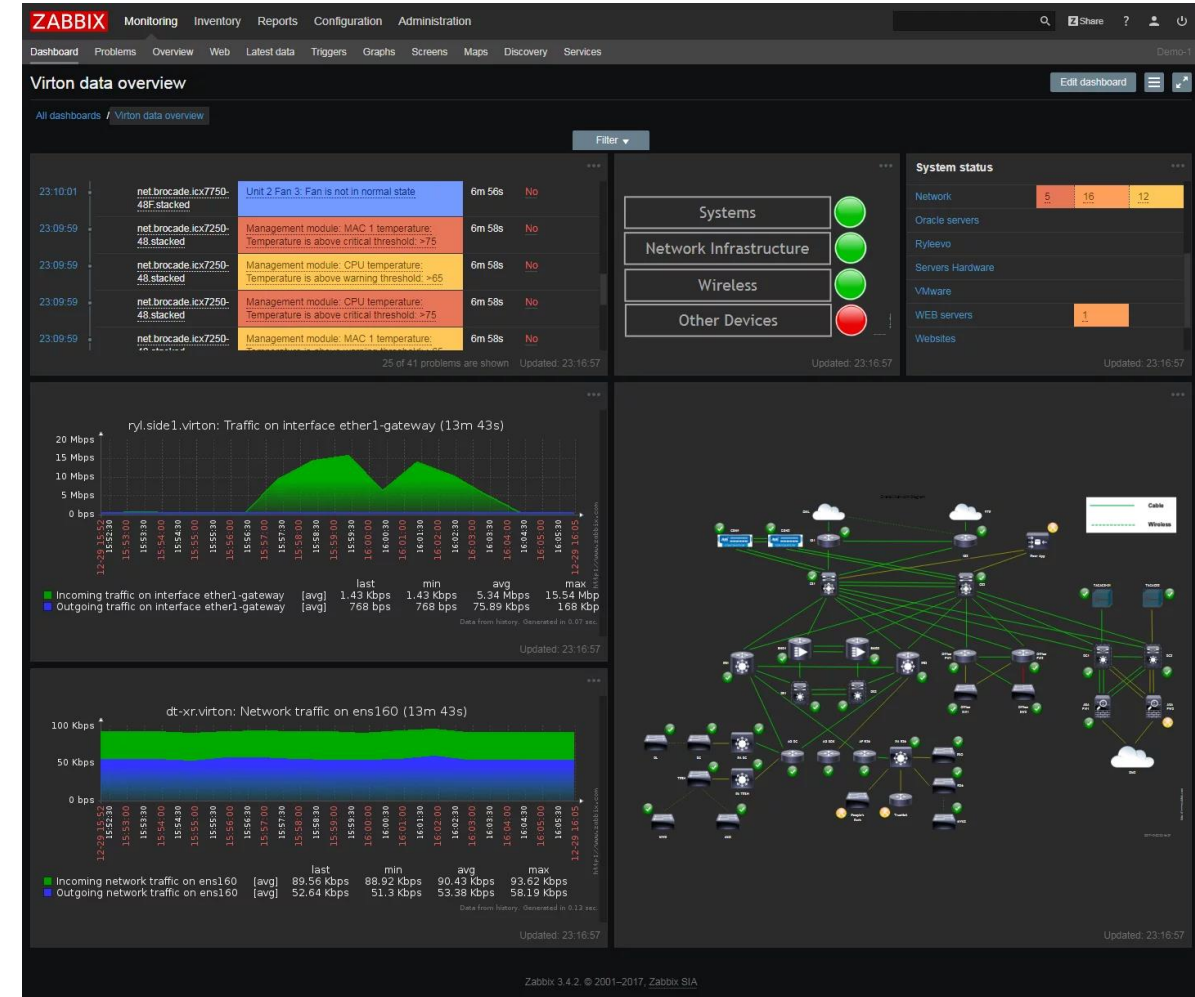
Popular monitoring software: ELK Stack

- Elastic Stack (ELK Stack): combines the capabilities of three open-source systems applicable to metrics collected from multiple sources
 - Elasticsearch: search and analytics
 - Logstash: inject and transform data from different sources, send to Elasticsearch
 - Kibana: visualization through charts and graphs based on data analyzed by Elasticsearch
- Integration via Metricbeat: correlates metrics from sources such as servers, Docker containers, Kubernetes, ...



Popular monitoring software: Zabbix

- One of the most popular open-source infrastructure monitoring tools for network, server, cloud, application, and databases
 - Provides visualisation
 - Supports multiple platforms (Windows, Linux, Unix, etc.) and gathers important metrics like CPU, memory, and network usage.
 - Lightweight agent with a small footprint, centrally managed from the Zabbix server
 - Easy integration with external applications via Zabbix API

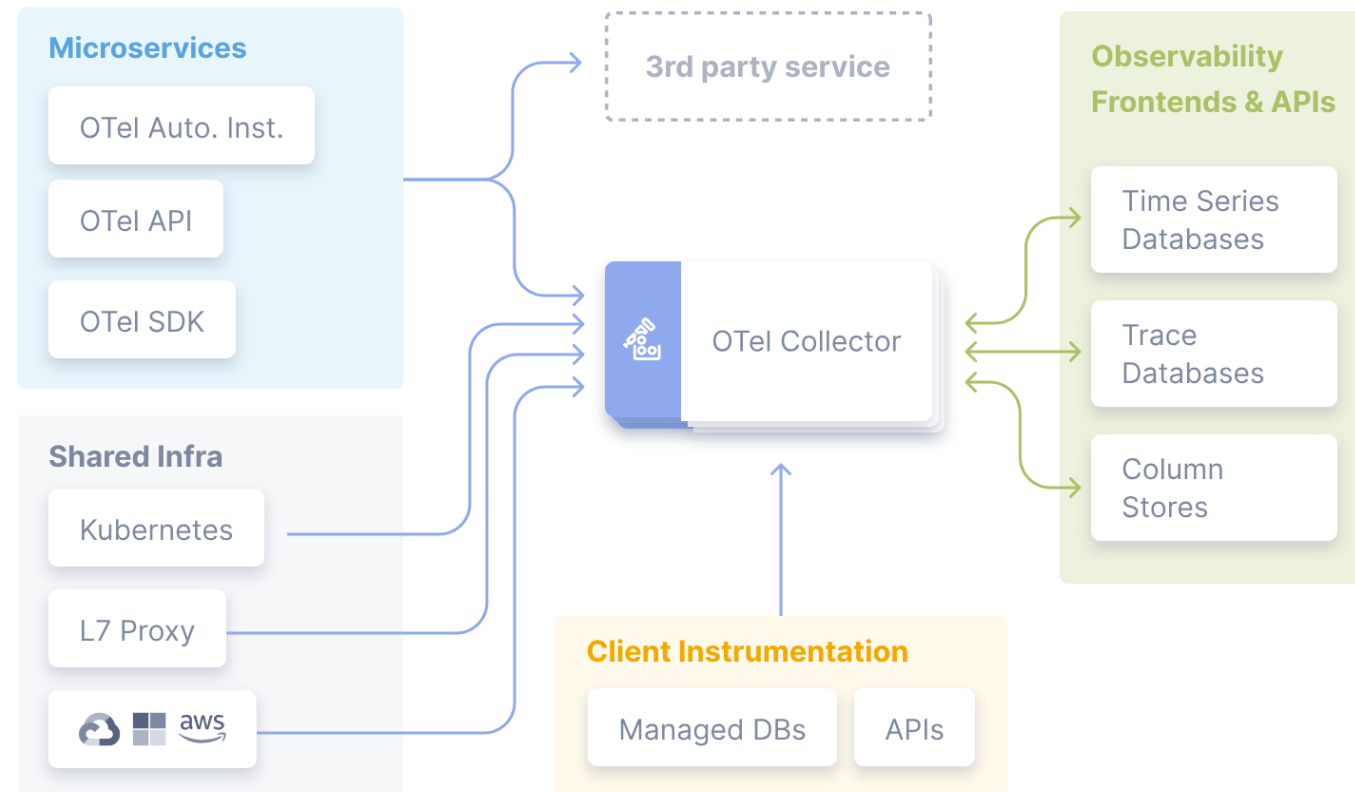


Other popular monitoring software

- Prometheus: open-source infrastructure monitoring tool created by former Google employees, originally intended to monitor heavily containerized environments.
- SolarWinds Server provides in-depth monitoring IT infrastructure, on-premises and in the cloud. Heavily involved in a security attack in 2020, see [US government data breach](#)
- Datadog and Dynatrace as major commercial solutions provide a comprehensive monitoring platform covering the infrastructure across cloud, on premises, and hybrid environments → thousands of out-of-the-box infrastructure metrics to view the health of application stack, containers, etc.
- Nagios: one of the oldest infrastructure monitoring tools available as open-source (Nagios Core) and enterprise solution (Nagios XI). Core is Linux based and can be extended through community-developed custom plugins.

Telemetry

- Telemetry = measurements at remote points and their transmission to receiving equipment for monitoring
- OpenTelemetry (OTel) = open-source observability framework with collection of tools, APIs, SDKs to instrument, generate, collect, and export telemetry data for analysis
- Cloud Native Computing Foundation (CNCF) incubating project → unified sets of vendor-agnostic libraries and APIs for collecting data and transferring it



<https://opentelemetry.io/docs/>

How does OpenTelemetry work?

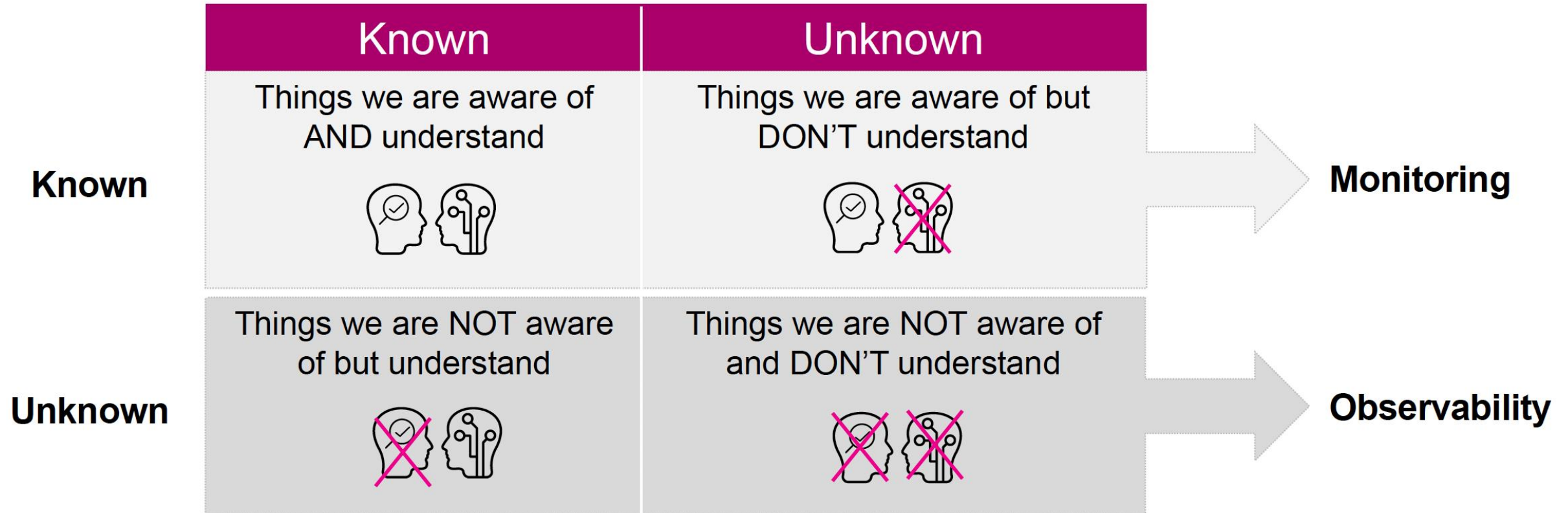
- OTel = specialized protocol for collecting telemetry data and exporting it to a target system
- Multiple steps for data lifecycle:
 - Instruments code with APIs, telling system components what metrics to gather and how to gather them
 - Pools the data using SDKs and transports it for processing and exporting
 - Breaks down the data, samples it, filters it to reduce noise or errors, and enriches it using multi-source contextualization
 - Converts and exports the data
 - Conducts more filtering in time-based batches, then moves the data onward to a predetermined backend
- Main difference to monitoring: Next to metric data also logs and traces collected

9.3 Observability

- Evidence-based debugging for complex systems is iterative process
 - Start with a high-level metric
 - Detangle based on fine-grained data/observations
 - Make the right deductions based on the evidence
- Good to know: Survival bias is the logical error of concentrating on the people or things that made it past some selection process, and overlooking those that did not, typically because of their lack of visibility
 - False conclusions in several different ways
 - Observability is based on the acquisition of data that allows to ask questions you didn't know you have and solve problems that you never thought of

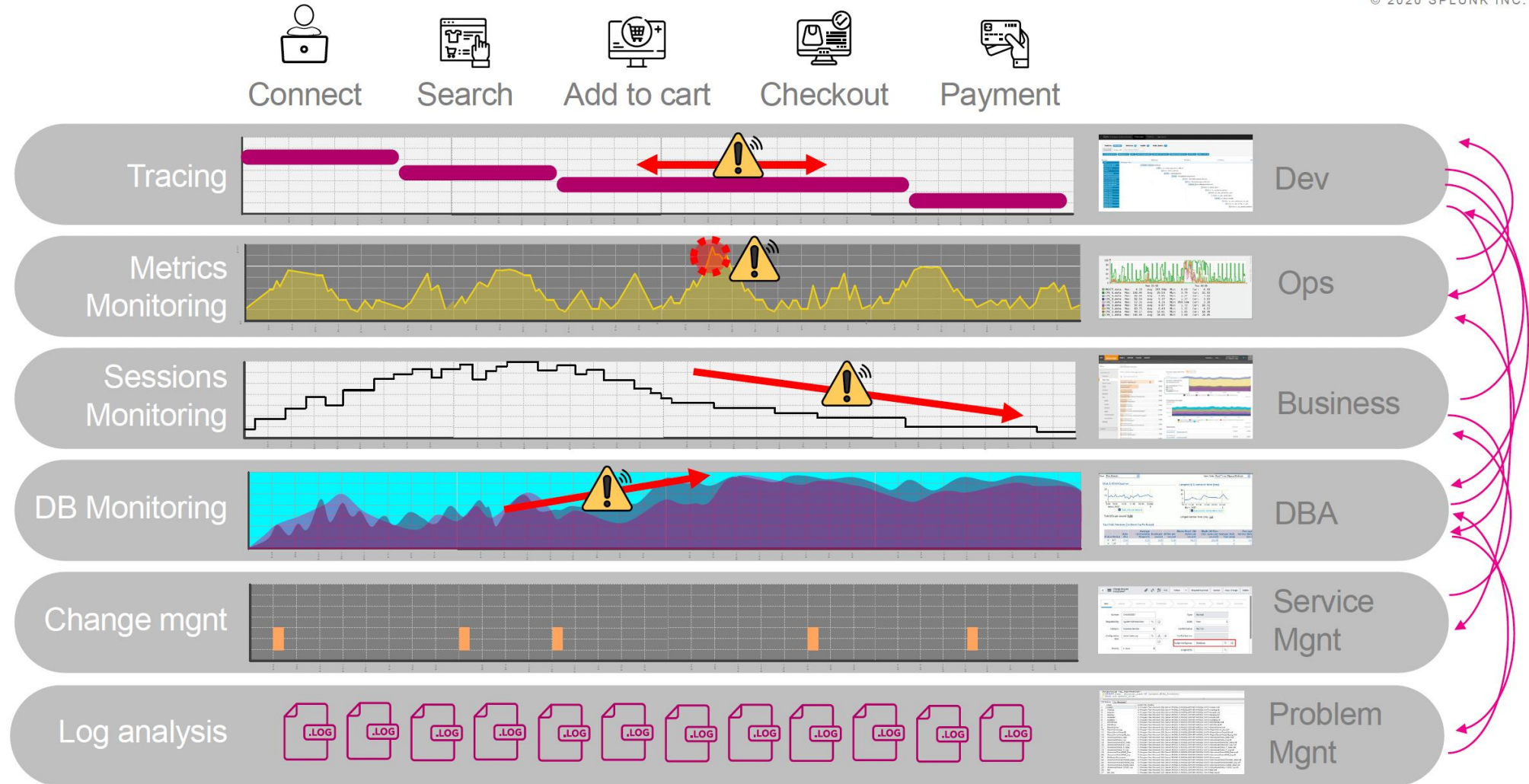


Today's knowns are yesterday's unknowns



Example@Splunk

© 2020 SPLUNK INC.

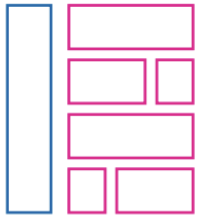


© ZUZU SPLUN



Cloud native drives need for observability

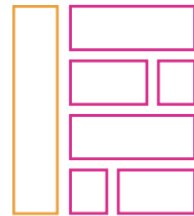
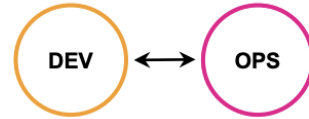
Retain & Optimize



Tightly Coupled Apps,
Slow Deployment Cycles



Lift & Shift



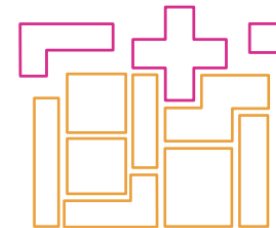
Primarily using
Cloud IaaS



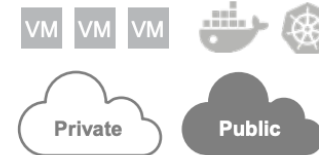
Re-Factor



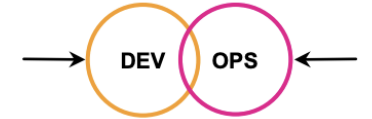
Cloud Managed e.g. RDS,
DynamoDB, SaaS



More Modular, but
Dependent App
Components



Re-Architect/ Cloud-Native



Cloud First Architecture



Loosely Coupled
Microservices, and
Serverless Functions



Definition Observability

- Observability measures how well the system's internal states can be inferred from knowledge of its external outputs
 - uses the data and insights that monitoring produces to provide a holistic understanding of the system, including its health and performance.
 - observability depends on how well the monitoring metrics can reflect and interpret system's performance indicators.
- Particular important property in microservices and containers due to the overall complexity



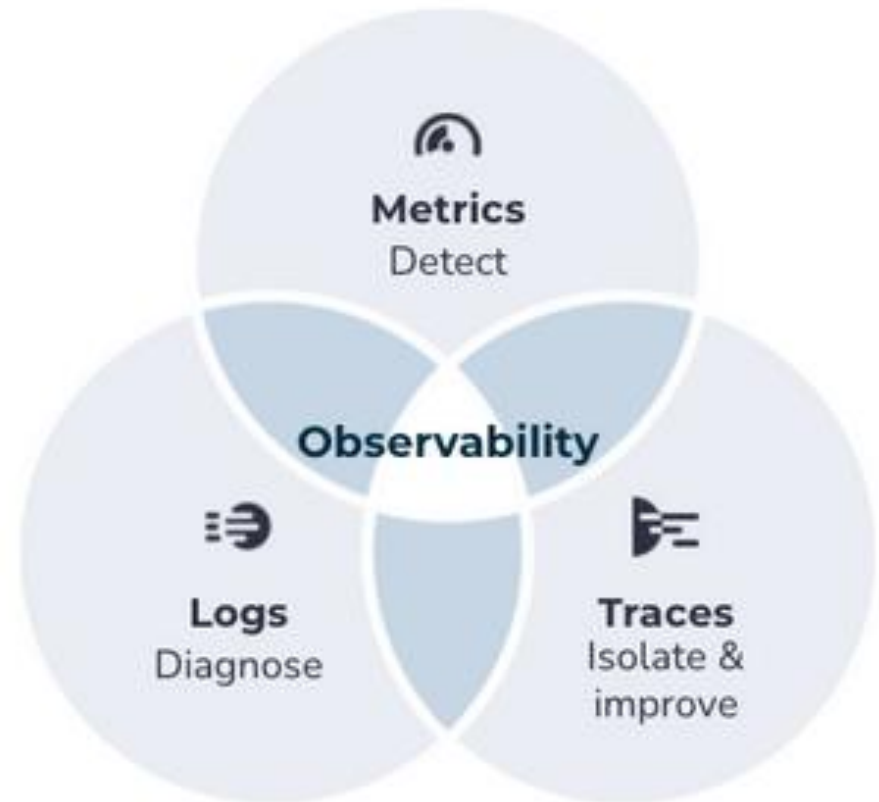
Uber microservice graph presented @ SREcon'23

Challenges of observability

- Accidental invisibility
 - Failing to properly filter or structure data sources that compete for attention can lead to accidental invisibility of important events and data. This can cause a critical condition to be missed because it's hidden from view or processing.
- Lack of source data
 - Not all important information is collected, particularly at the application level and as it relates to tracing of workflows. Unlike resource or component status, traces of workflows usually require special software modifications to enable.
- Multiple information format
 - It can be difficult to assemble the right information and interpret what's available when the same type of data comes in different formats from different sources.

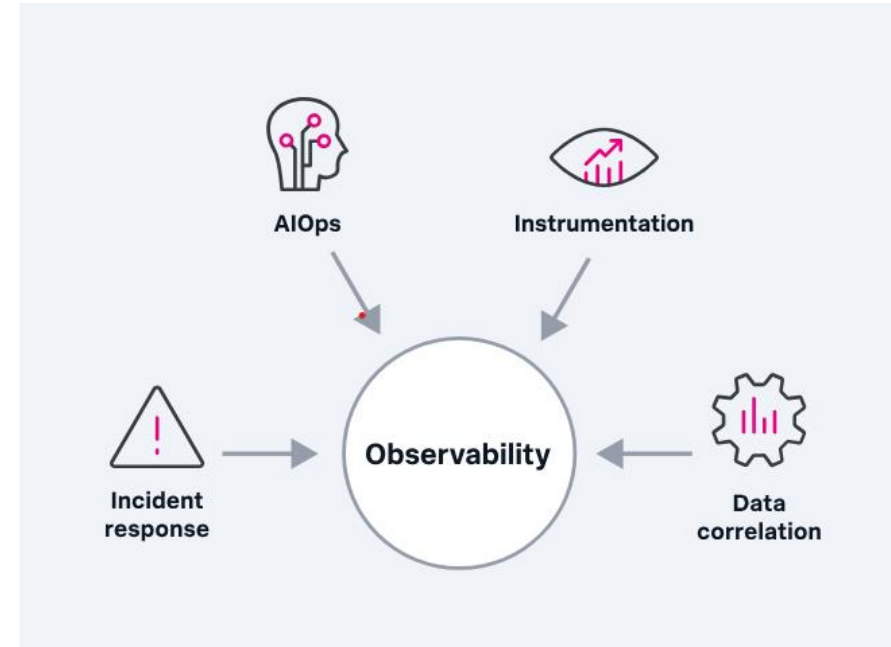
The three pillars of observability

- Combining metrics, logs, and traces for observability is the only way to understand complex environments
 - Metrics tell us the “what”
 - Logs tell us the “why”
 - Traces tell us the “where”
- A good observability solution correlates the data from multiple sources and visualizes the key aspects



Implementing Observability

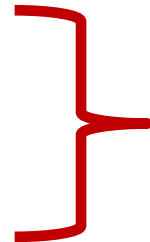
- Typically, four components involved
 - Instrumentation: Measuring tools that collect telemetry data from a container, service, application, host, enabling visibility across infrastructure.
 - Data correlation: Telemetry data is processed and correlated to create context and enable automated or custom data curation for time series visualizations.
 - Incident response: get data about outages to the right people and teams based on on-call schedules and technical skills.
 - AIOps: ML models to automatically aggregate, correlate and prioritize incident data, to filter alert noise, detect issues, accelerate incident response when they do.
- Focus here: Log analytics and AIOps



https://www.splunk.com/en_us/blog/learn/observability.html

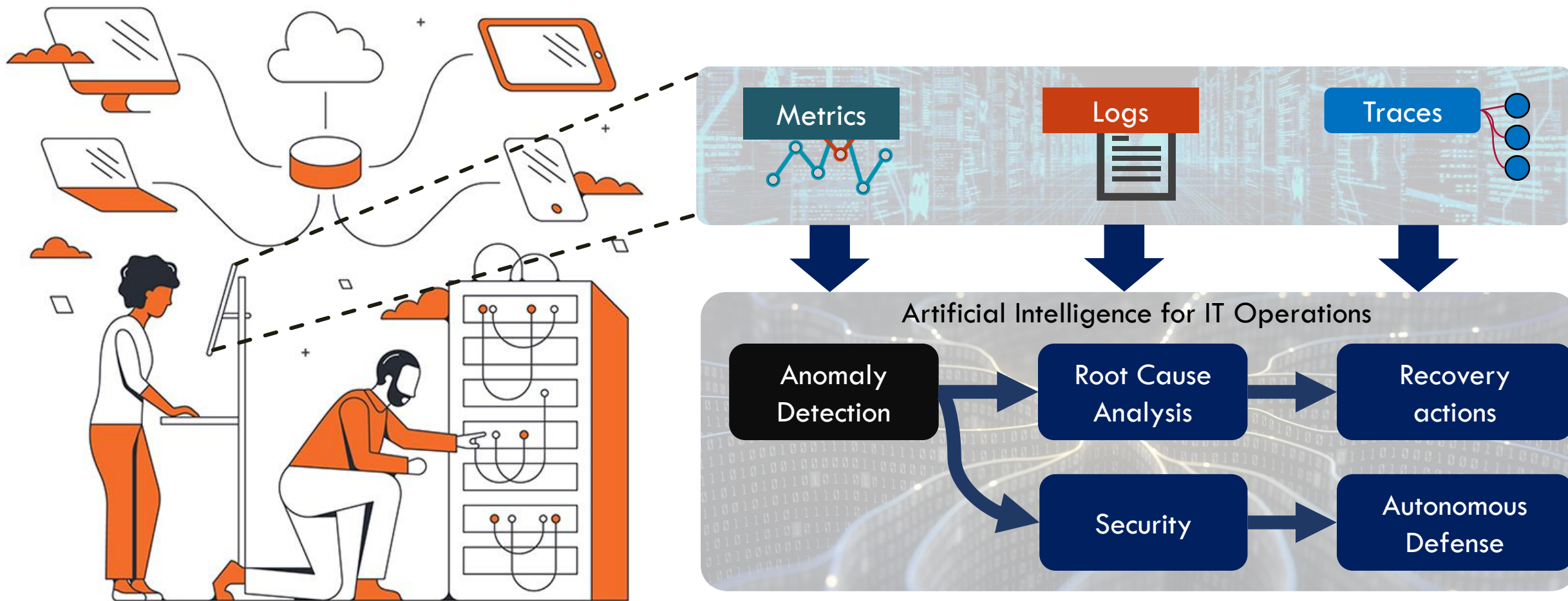
9.4 AIOPs (AI-enabled IT Operations)

- Introduce additional intelligence in operations: NREs, SREs, AI-based tools
- Automate operations
 - Capacity planning
 - Balancing & Resource Utilization
 - Storage Management
 - Threat Detection & Analysis
 - Predictable Fault Tolerance



Key for Cloud Resilience

AIOPs as Key for Reliability



AI Ops: Where we go?

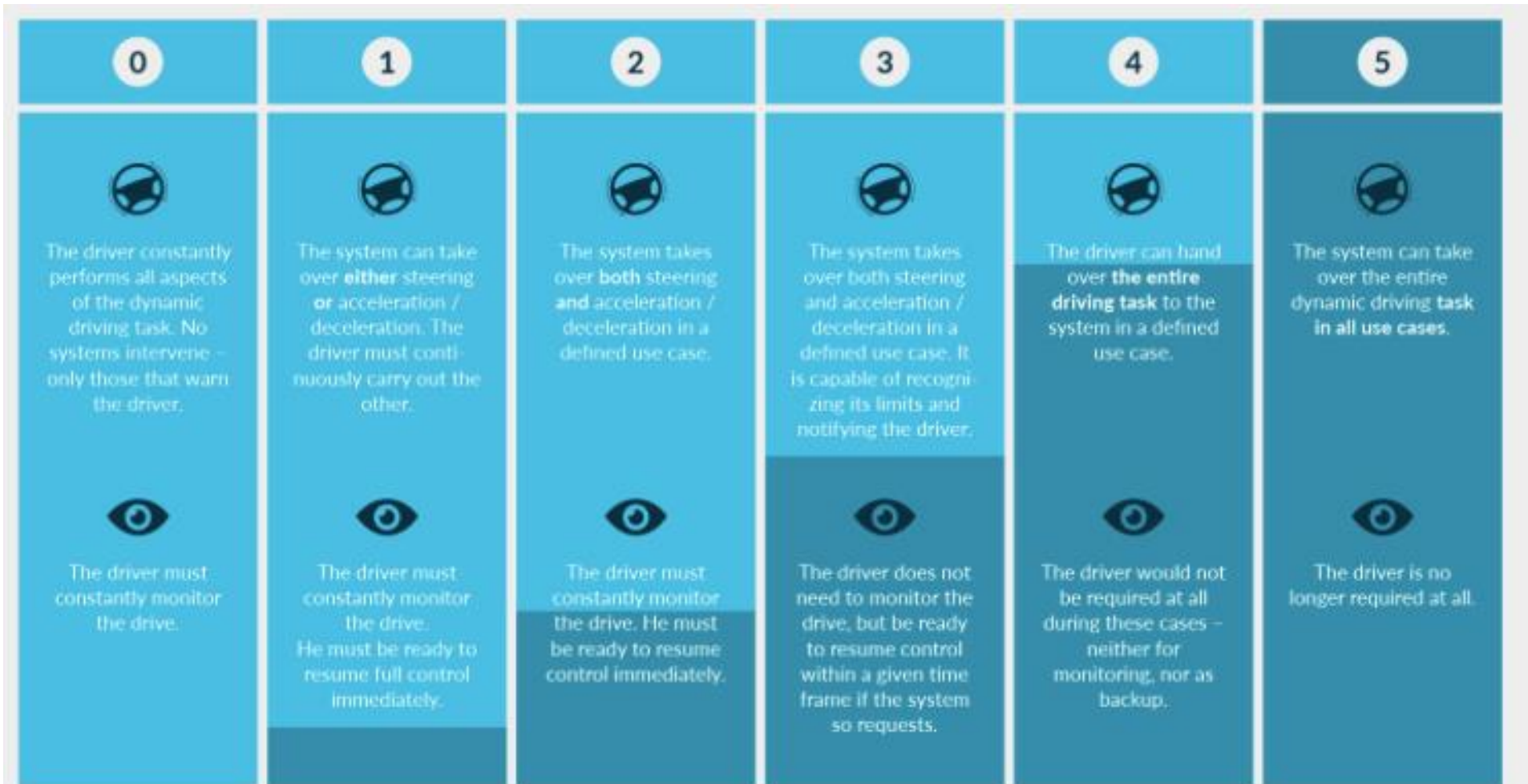


People in the booths are observers, no actors. Action by AI in background, latency crucial!

Automation in self-driving vehicles

Driver does it all

No driver needed



www.2025ad.com/latest/the-levels-of-automation/

AI-Governance: How far do you want to go?

Level 1: Human operator decides, computer implements

Level 2: AI determines options, human selects feasible option

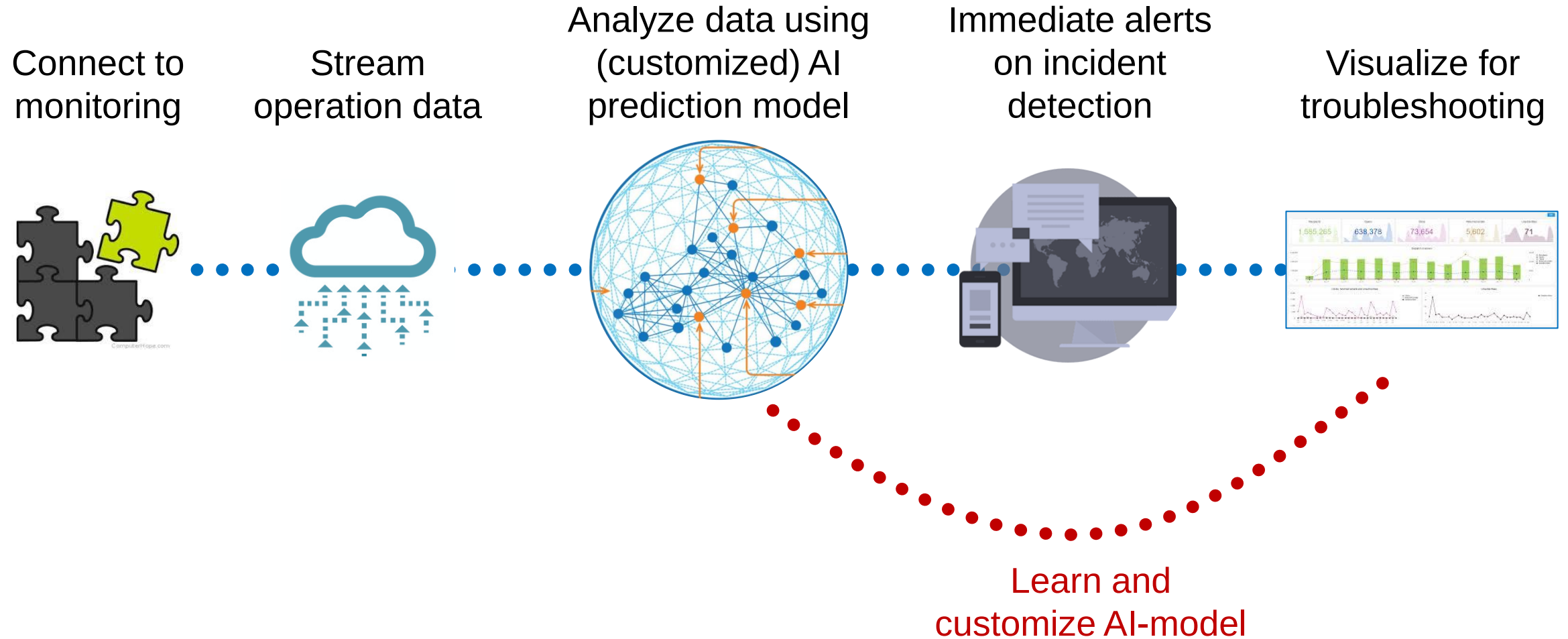
Level 3: AI selects, implements action after operator's approval (*e.g. automatic server and consumer device updates*)

Level 4: AI selects, initiates action, informs operator if rollback wanted (*e.g. runtime add/remove nodes, replicate DB, re-size partitions*)

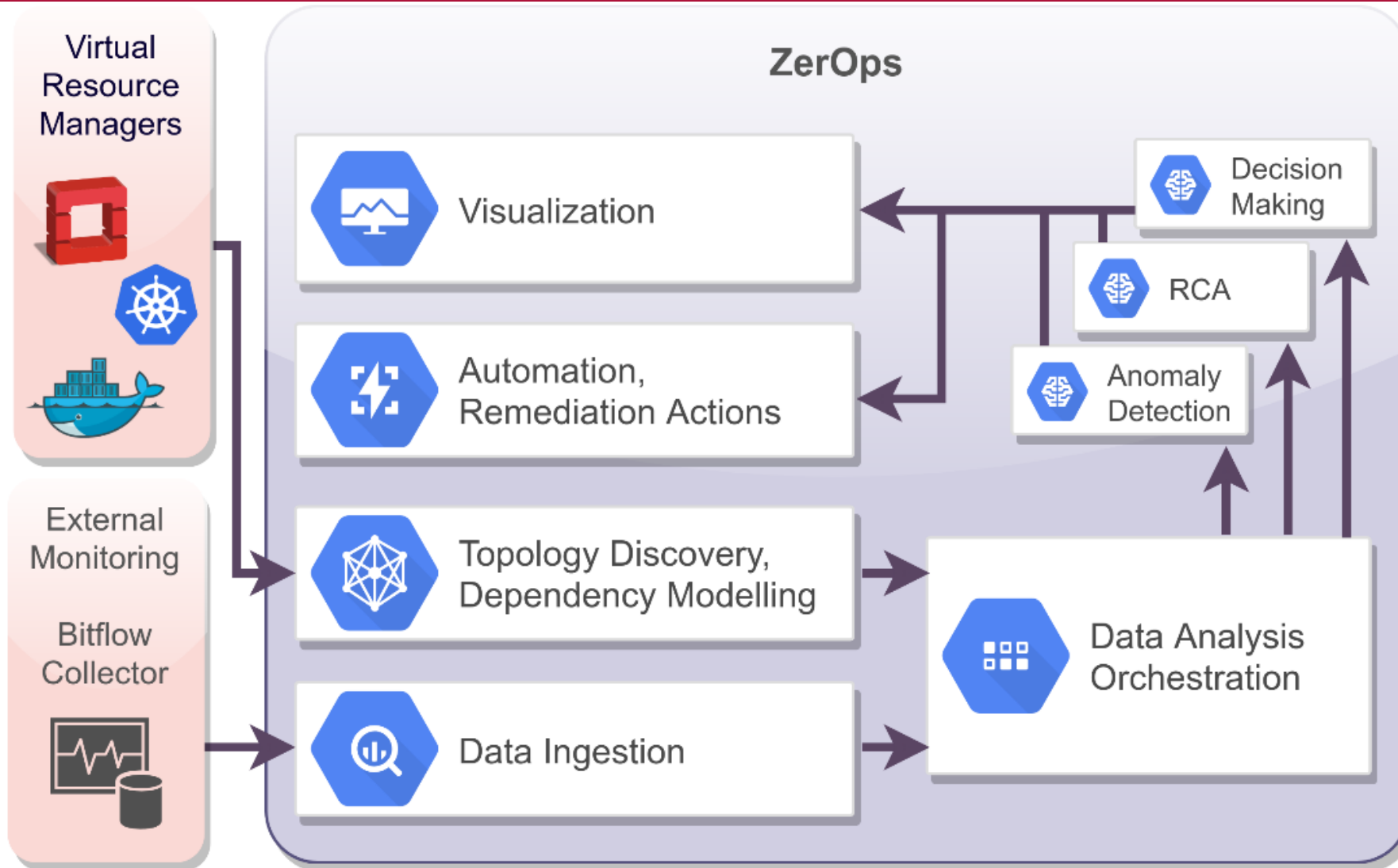
Level 5: AI determines malfunction of applied option, selects new option from pre-defined activities (*e.g. automatic remediation for availability*)

Level 6: AI determines malfunction of applied option, repeats the process by creating new remediation activity (*No systems yet*)

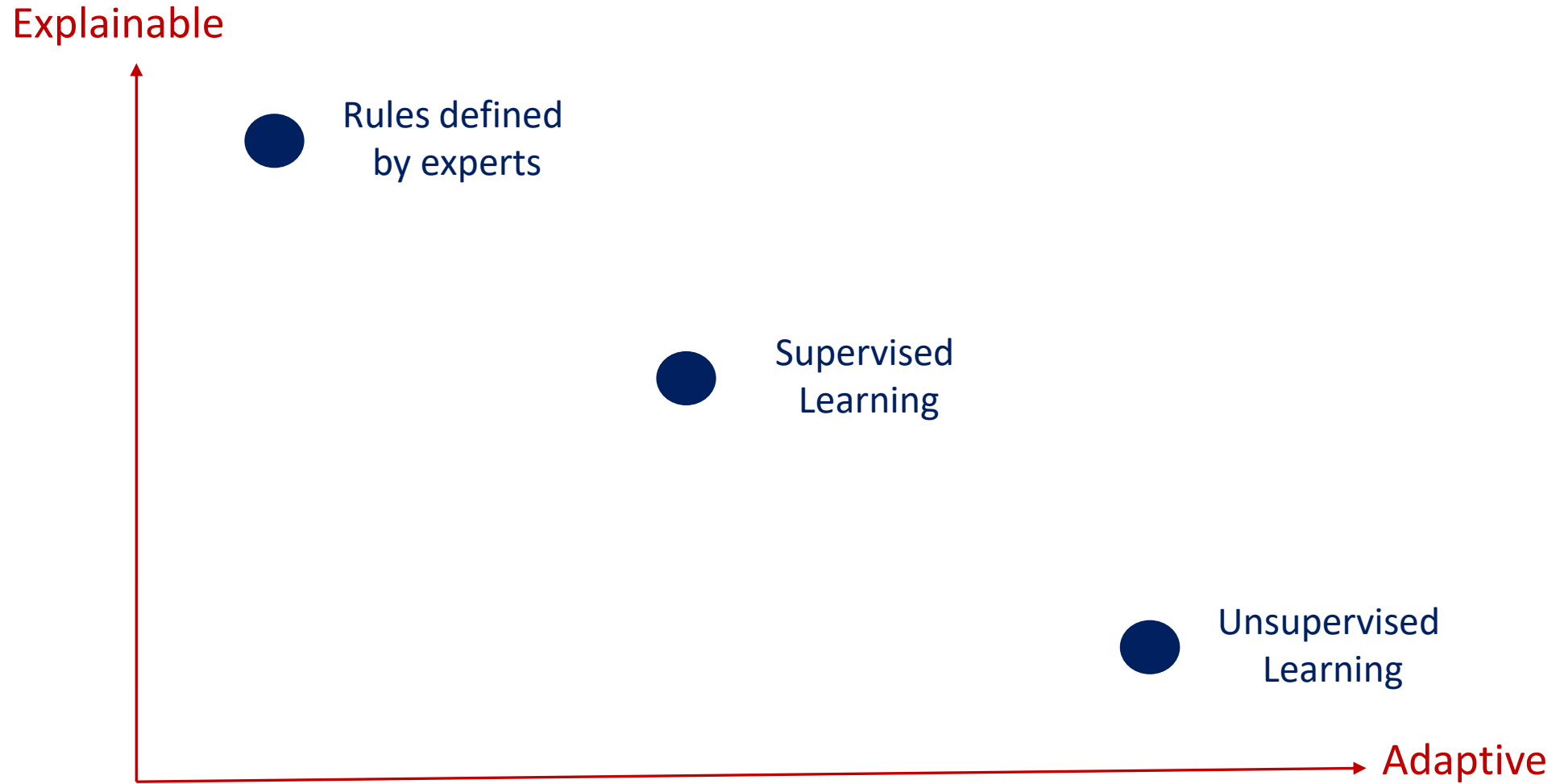
Workflow



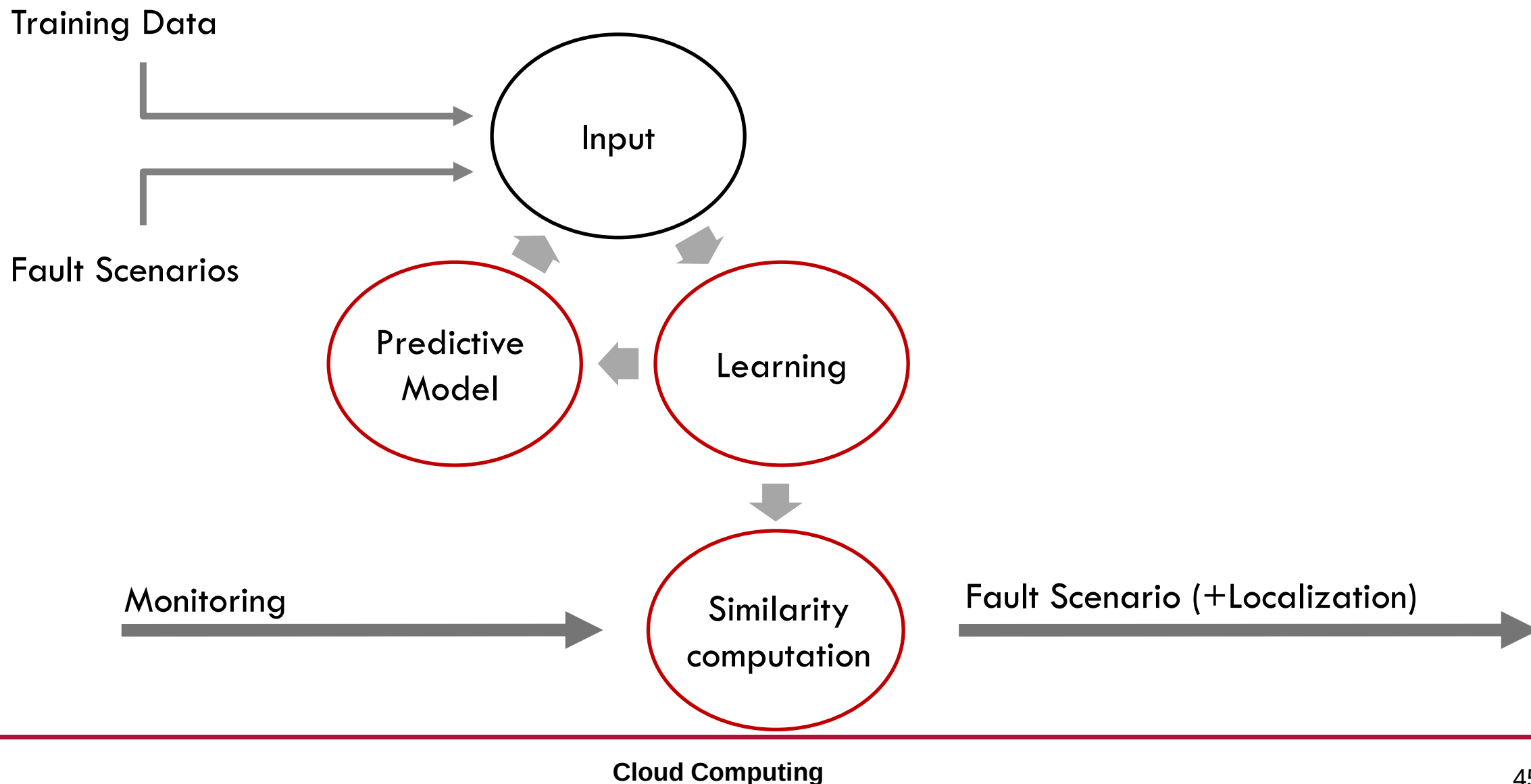
AIOPs platforms



AIOPs approaches

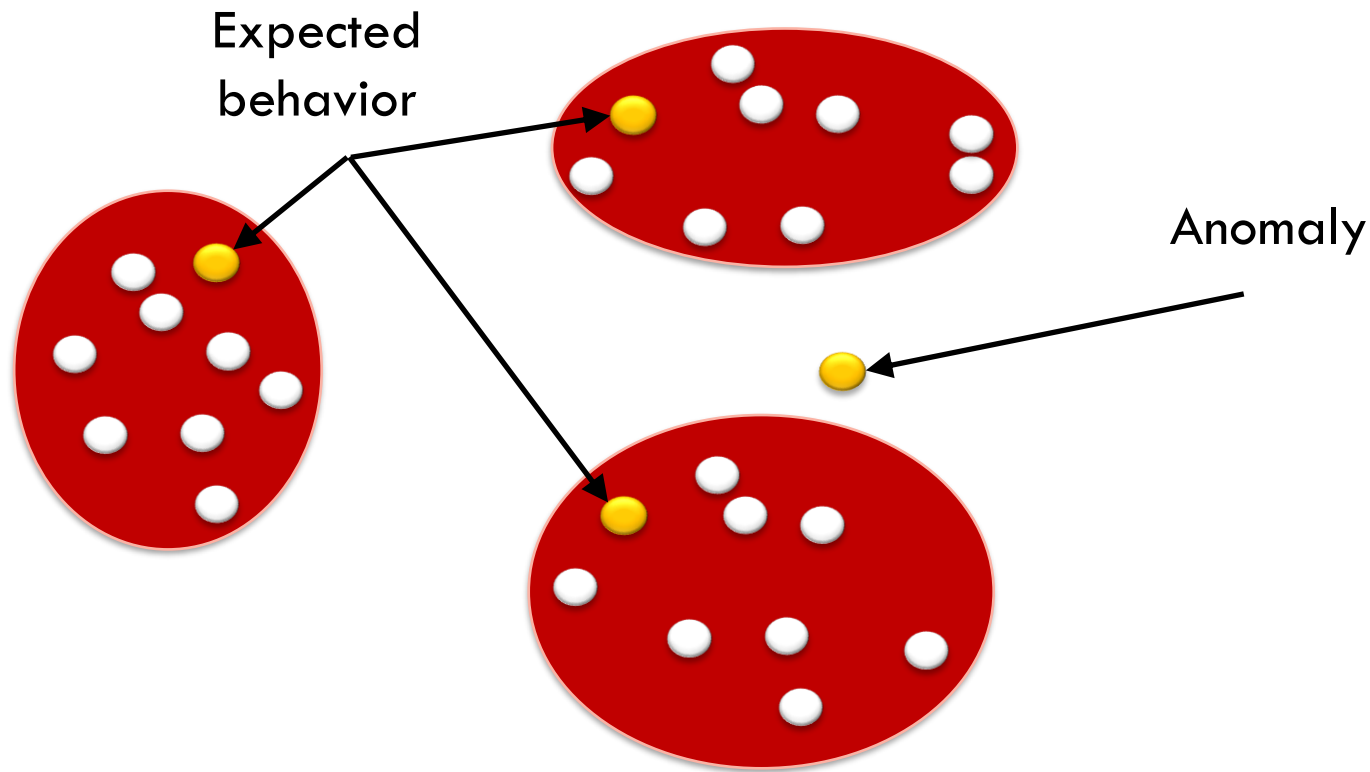


Supervised Learning on metric data



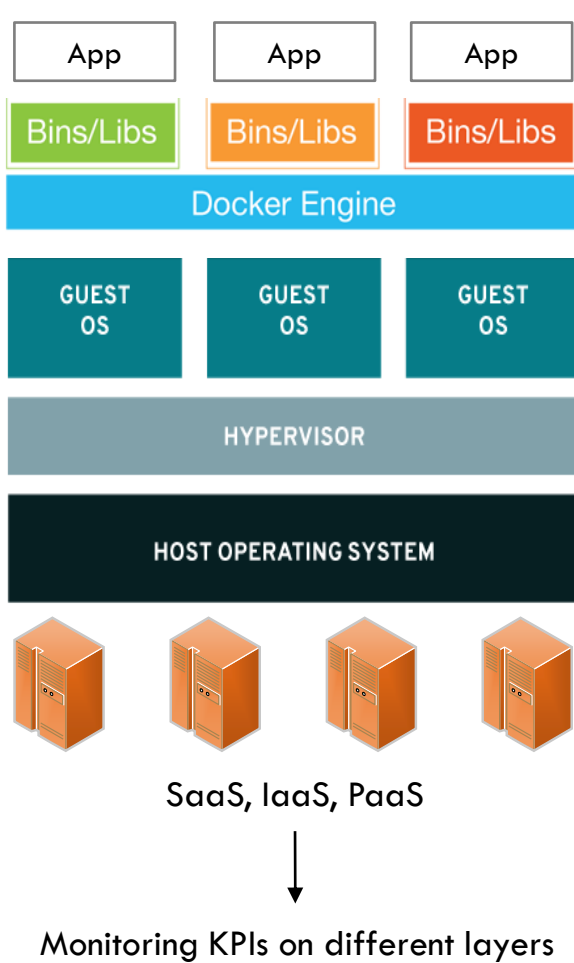
Unsupervised Learning on metric data

- Learn normal system state and identify deviations as anomalies
- Which state is normal? $\Rightarrow$ Needs context, cross-layer understanding, specific data

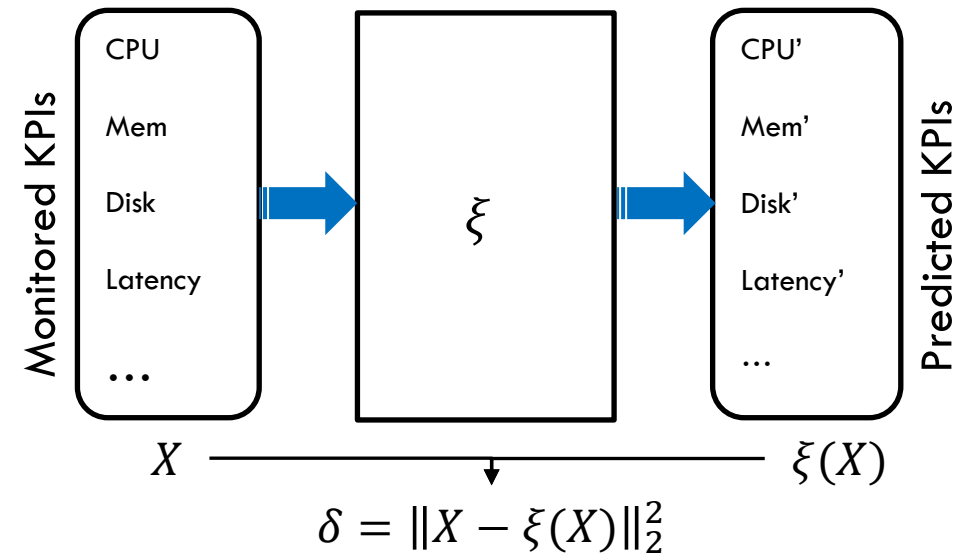


No predefined patterns, #normal samples $\gg$ #anomaly samples

IFTM: Unsupervised AD



Identity Function: Identity value prediction

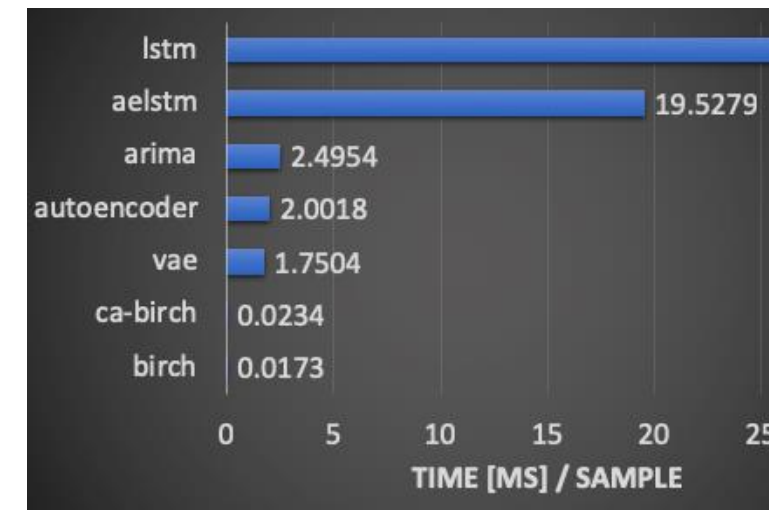


Threshold Model: Decision boundary τ for δ

Normal: $\delta \leq \tau$

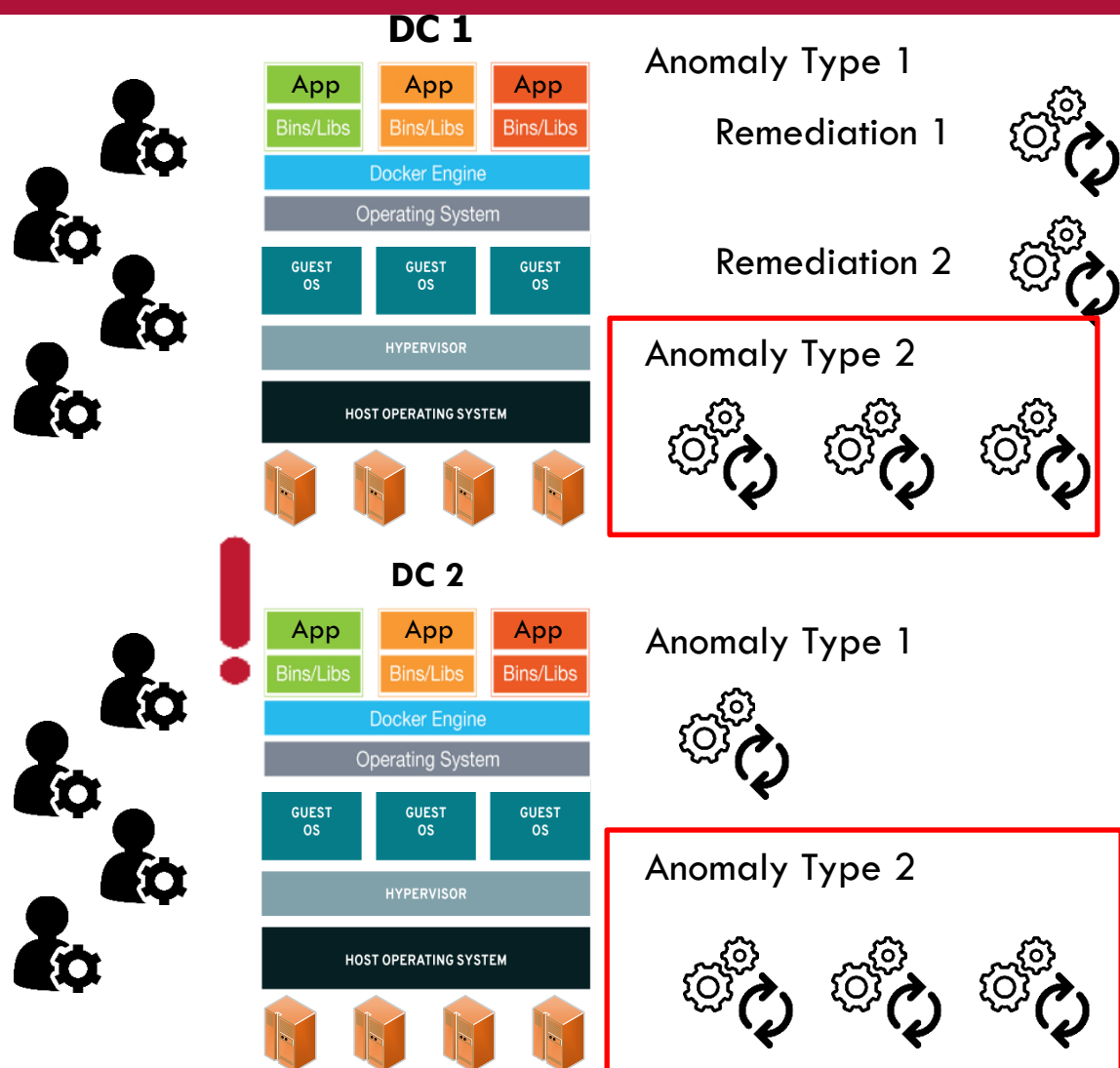
Anomaly: $\delta > \tau$

Overhead **BIRCH** = **B**alanced
Iterative **R**educing and **C**lustering
using **H**ierarchies



ACC=0.91

Remediation Action Selection

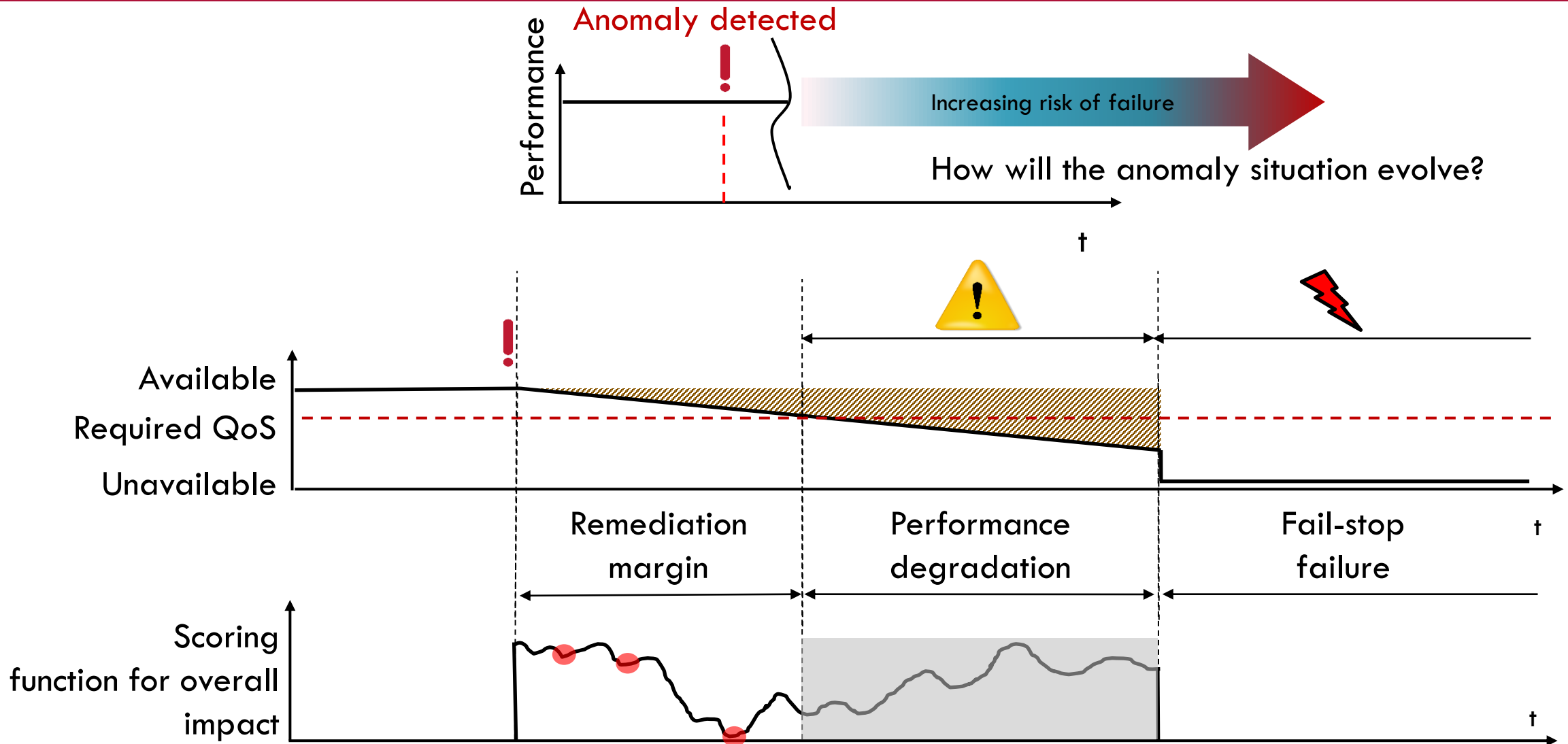


Find and apply the best remediation workflow

MCDA Methods (e.g. weighted sum model)

	$n(c_1)$	$n(c_2)$	...	$n(c_k)$	S
A_1	0.2	0.6	...	0.4	Σ_1
A_2	0.9	0.2	...	0.77	Σ_2
...	...	...		...	...
A_n	0.3	0.1	...	0.91	Σ_n
W	w_1	w_2		w_k	

Remediation Scheduling



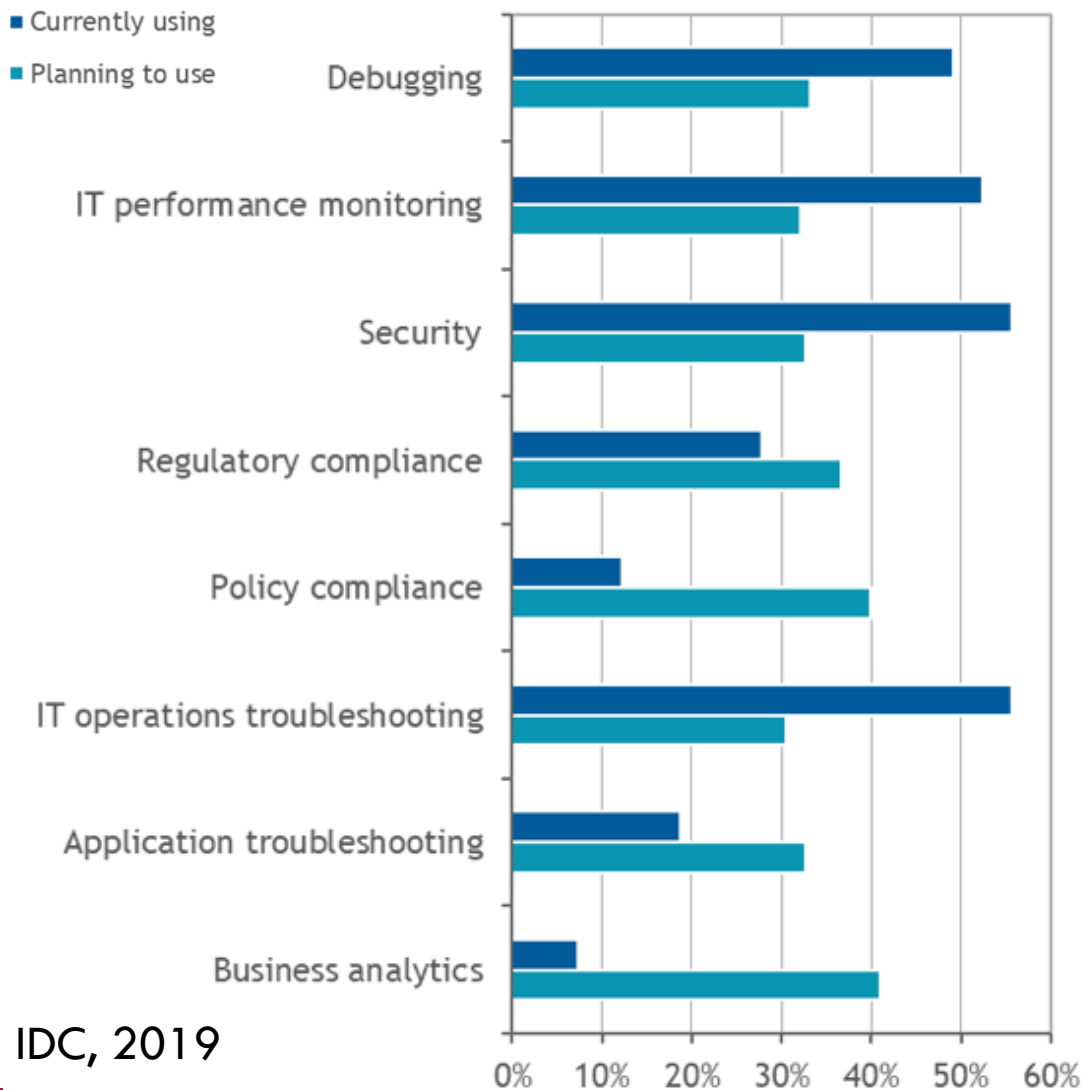
9.5 AIOPs using log data

```
081111 083419 24621 INFO dfs.DataNode$DataXceiver: Receiving block blk_5214640714119373081 src:
/10.251.121.224:47915 dest: /10.251.121.224:50010
081111 083419 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock:
/user/root/rand7/_temporary/_task_200811101024_0014_m_001575_0/part-01575. blk_5214640714119373081
081111 083420 24633 INFO dfs.DataNode$DataXceiver: Receiving block blk_5214640714119373081 src:
/10.251.121.224:57800 dest: /10.251.121.224:50010
081111 083422 24621 INFO dfs.DataNode$DataXceiver: writeBlock blk_5214640714119373081 received
exception java.io.IOException: Could not read from stream
081111 104136 26436 INFO dfs.DataNode$DataXceiver: Receiving block blk_-3208483482800741142 src:
/10.251.111.209:34510 dest: /10.251.111.209:50010
081111 104136 26954 INFO dfs.DataNode$DataXceiver: Receiving block blk_-3208483482800741142 src:
/10.251.203.80:46033 dest: /10.251.203.80:50010
081111 104136 27196 INFO dfs.DataNode$DataXceiver: Receiving block blk_-3208483482800741142 src:
/10.251.111.209:46712 dest: /10.251.111.209:50010
081111 104136 35 INFO dfs.FSNamesystem: BLOCK* NameSystem.allocateBlock:
/user/root/randtxt9/_temporary/_task_20 0811101024_0016_m_001470_0/part-01470. blk_-
3208483482800741142
081111 104233 26437 INFO dfs.DataNode$PacketResponder: PacketResponder 1 for block blk_-
3208483482800741142 terminating
.....
```

Automatically detected anomaly

Logs are Major Troubleshooting Data

DevOps use logs to repair IT failures



IDC, 2019



Log Search

(70% companies choose this)

- Cheap, but requires a lot of manual work
- Does not scale to complex systems
- A lot of guesswork

From log analysis to log analytics

Log analysis Clustering + Time Series



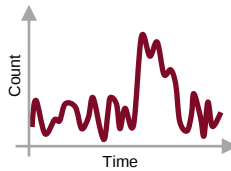
1. Unstructured

```
Dec 10 06:55:46: Invalid user webmaster from 173
Dec 10 06:55:46: input_userauth_request: invalid u
Dec 10 06:55:46: pam_unix(sshd:auth): check pas
Dec 10 06:55:48: Connection closed by 173.234.31
Dec 10 07:02:47: Connection closed by 212.47.254
Dec 10 07:07:38: Invalid user test9 from 52.80.34.1
Dec 10 07:07:38: input_userauth_request: invalid i
Dec 10 07:07:38: pam_unix(sshd:auth): check pas
Dec 10 07:07:45: Failed password for invalid user i
Dec 10 07:07:45: Received disconnect from 52.80.
Dec 10 07:08:28: Invalid user webmaster from 173
Dec 10 07:08:28: input_userauth_request: invalid i
Dec 10 07:08:28: pam_unix(sshd:auth): check pas
Dec 10 07:08:30: Connection closed by 173.234.31
Dec 10 07:11:42: Invalid user chen from 202.100.1
```

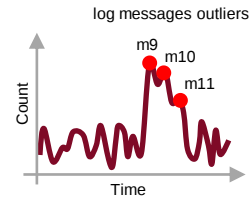
2. Clustering

```
Dec 10 06:55:46: Invalid user webmaster from 173
Dec 10 06:55:46: input_userauth_request: invalid i
Dec 10 06:55:46: pam_unix(sshd:auth): check pas
Dec 10 06:55:48: Connection closed by 173.234.31
Dec 10 07:02:47: Connection closed by 212.47.254
Dec 10 07:07:38: Invalid user test9 from 52.80.34.1
Dec 10 07:07:38: input_userauth_request: invalid i
Dec 10 07:07:38: pam_unix(sshd:auth): check pas
Dec 10 07:07:45: Failed password for invalid user i
Dec 10 07:07:45: Failed password for invalid user i
Dec 10 07:08:15: Failed password for invalid user i
Dec 10 07:08:28: input_userauth_request: invalid i
Dec 10 07:08:28: pam_unix(sshd:auth): check pas
Dec 10 07:08:30: Connection closed by 173.234.31
Dec 10 07:11:42: Invalid user chen from 202.100.1
```

3. Time series



4. Anomaly detection

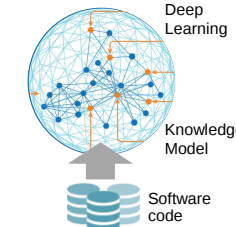


- Simple, but expensive (DevOps/SRE hours)
- High potential to overlook errors
- Human fatigue

AI-driven log analytics Deep Learning + Failure models



0. AI Model



1. Unstructured

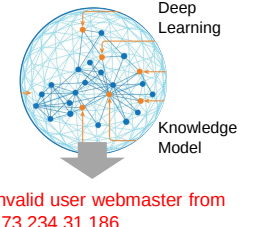
```
Dec 10 06:55:46: Invalid user webmaster from 173
Dec 10 06:55:46: input_userauth_request: invalid i
Dec 10 06:55:46: pam_unix(sshd:auth): check pas
Dec 10 06:55:48: Connection closed by 173.234.31
Dec 10 07:02:47: Connection closed by 212.47.254
Dec 10 07:07:38: Invalid user test9 from 52.80.34.1
Dec 10 07:07:38: input_userauth_request: invalid i
Dec 10 07:07:38: pam_unix(sshd:auth): check pas
Dec 10 07:07:45: Failed password for invalid user i
Dec 10 07:07:45: Received disconnect from 52.80.
Dec 10 07:08:28: Invalid user webmaster from 173
Dec 10 07:08:28: input_userauth_request: invalid i
Dec 10 07:08:28: pam_unix(sshd:auth): check pas
Dec 10 07:08:30: Connection closed by 173.234.31
Dec 10 07:11:42: Invalid user chen from 202.100.1
```

2. Structured

```
{
  "requestMethod": "GET",
  "times": "2020-10-12T07:20:50.52Z",
  "logging.googleapis.com/insertId": "42",
  "logging.googleapis.com/labels": {
    "user_label_1": "value_1",
    "user_label_2": "value_2"
  },
  "logging.googleapis.com/operation": {
    "id": "get_data",
    "producer": "github.com/MyProject/MyApplication",
    "first": "true"
  },
  "logging.googleapis.com/sourceLocation": {

```

3. AI Inference



- Fast and efficient
- Challenging definition of AI models
- Dealing with false-positives

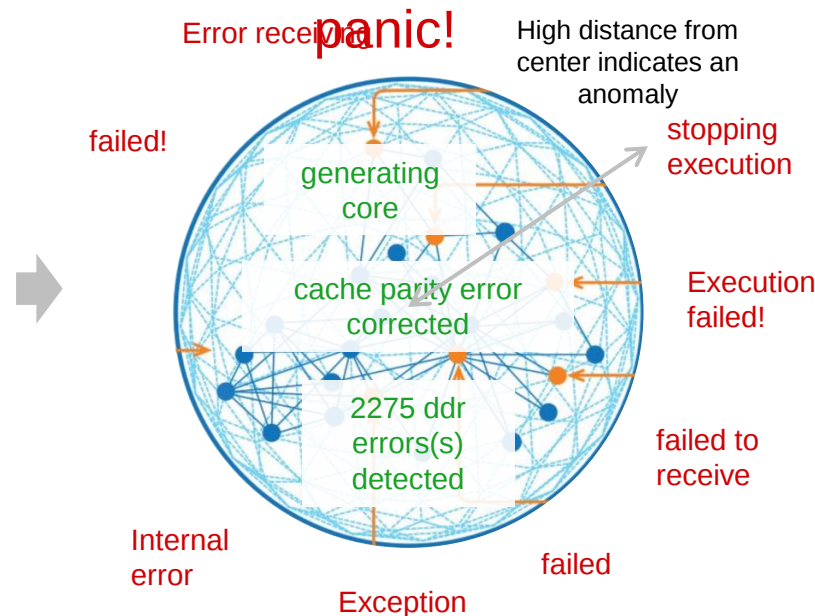
Deep Learning for Log Anomaly Detection

Learning

From millions lines of code and logs
(*GitHub, USENIX, CFDR, ...*)

```
ERROR rts panic! - stopping execution
ERROR Error receiving packet on tree network,
      expecting type 57 instead of type 3
FAILURE FgBioMain.scala Execution failed!
ERROR YTDL has failed, everyone panic
ERROR Internal error. Packet loop has exited.
ERROR Exception in packet out
FAILURE NFQ failed to receive a packet
```

```
INFO Instruction cache parity error corrected
      generating core.
INFO 2275 ddr errors(s) detected and corrected
      on rank 0, symbol 17, bit 1
```

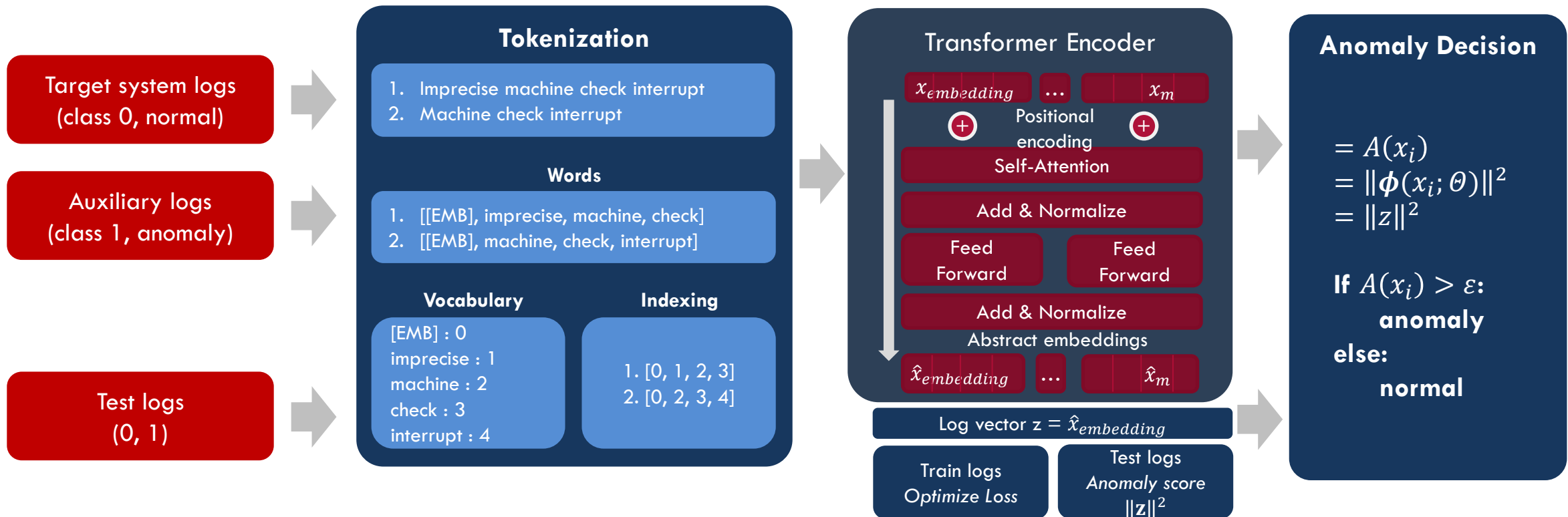


Detecting Customer logs

1. Service entered panic mode!
2. Error receiving packet on tree network
3. Network caused internal error
4. Nova connected to neutron
5. Plugin loaded in 5 seconds

Log Anomaly Detection: Model architecture

Input data → Preprocessing → Learning → Prediction



Use Case:

Log search vs. AI-driven log analytics

~350.000 log lines per day (REAL data)
60 types of errors

Log search
240 minutes

- **Experienced DevOps** conduct troubleshooting
- Reading logs and searching for **hints**
- **Search filters**: grep, awk ... and error, warning ...



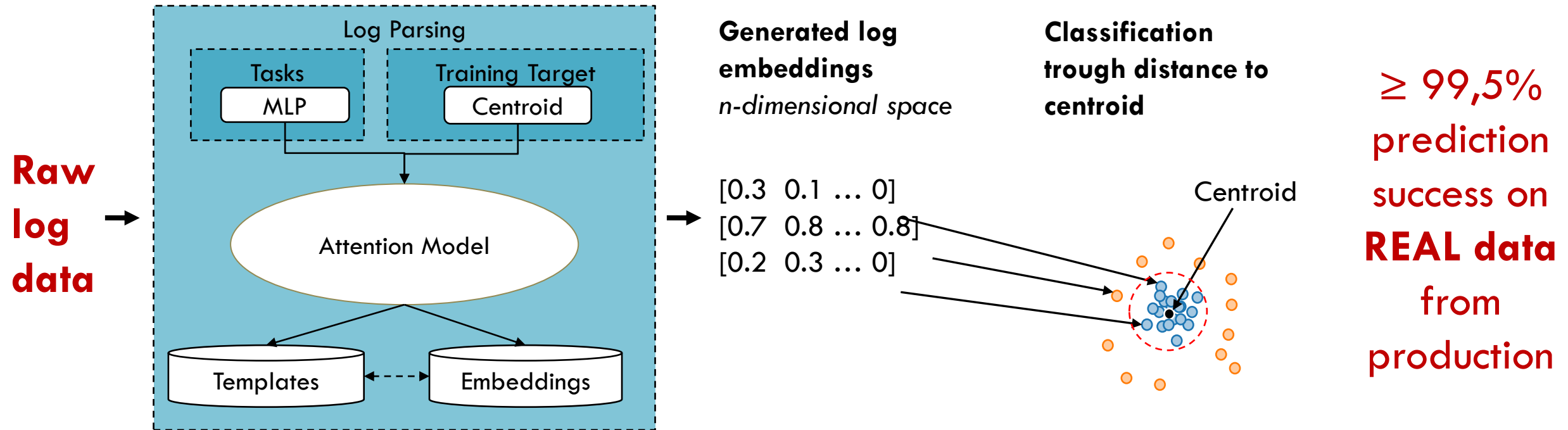
AI-driven log analytics
4 minutes

- **59s** uploading ~350.000 log lines
- **64s** pre-processing
- **90s** AI-driven log analytics

Use Case:

Disk Failure Prediction Through Logs

- Detect abnormal log lines in real SSD dataset (from large company)
- Classifies raw log line as normal or abnormal with one single model



Logging

AUTO LOGGING

⊕ 4 auto-logs added
⊖ 2 removed states
📄 70 % quality
⚙️ 4 levels corrected

Coverage
30%
7 blocks for revision

VERIFICATION

⚠️ 35 % risk
⚠️ 2 faults
⊕ 3 added states
↔️ 1 freq. change
🔄 4 recurring states

Quality Gate
Failed
1 condition failed

INCIDENT DETECTION

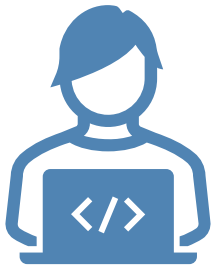
⚠️ 0 incidents
📄 6K logs
⊗ 0 semantic threats
⊗ 12 level threats

Monitoring
Passed
No incidents

Ensure log quality and coverage

Continuous verification in CI/CD

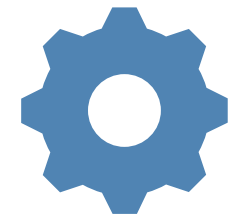
Predict & detect incidents



Code



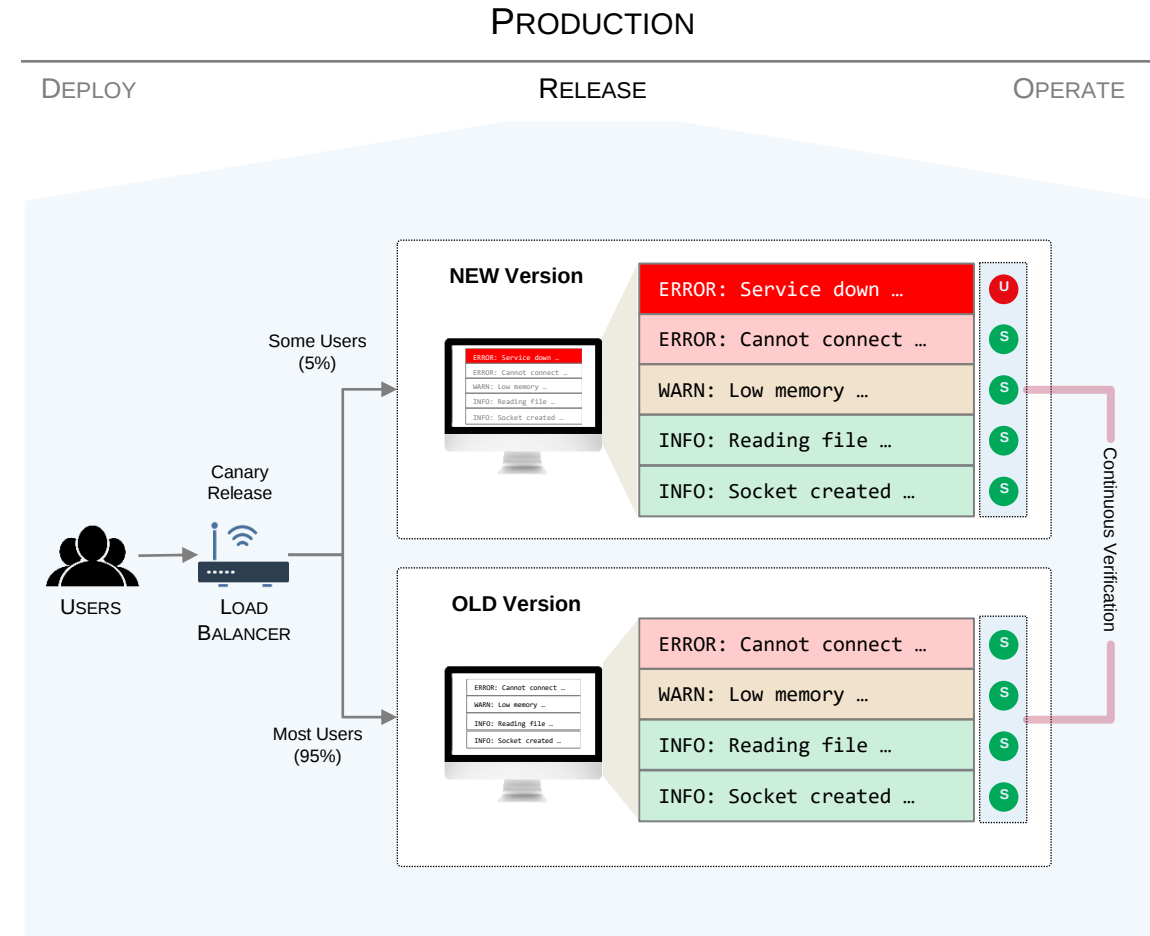
Verify



Deploy

Testing in Production without Writing Tests

- Test updates on live user traffic rather than in staging environment
 - do not perturbate user experience
 - non-intrusive techniques required
- Run NEW and OLD side by side
 - if OLD and NEW generate “similar” logs for a subset of requests, NEW can be classified as bug-free
 - Unseen messages analysed semantically to determine the risk for the release
 - If above threshold => block the release



Integration in CI/CD

Development



Line 12	ERROR: Service down ...
Line 34	ERROR: Cannot connect ...
Line 2	WARN: Low memory ...
Line 24	INFO: Reading file ...
Line 26	INFO: Socket created ...
Line 65	ERROR: SSL failure
Line 3	WARN: I/O parity error

Staging



CALL TEST
PUSH Logs
with
tag=test_id

TEST 1	ERROR: Service down ...
TEST 2	ERROR: Cannot connect ...
	WARN: Low memory ...
TEST 3	INFO: Reading file ...
	INFO: Socket created ...
	ERROR: SSL failure
	WARN: I/O parity error

Test-Log
Coverage
70%



Production Release

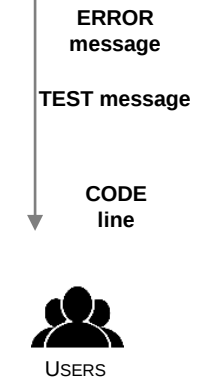
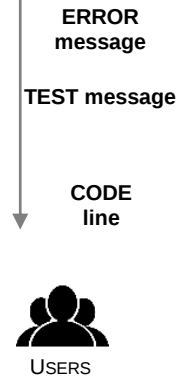


ERROR: Cannot connect ...	\$
WARN: Low memory ...	\$
INFO: Reading file ...	\$
INFO: Socket created ...	\$

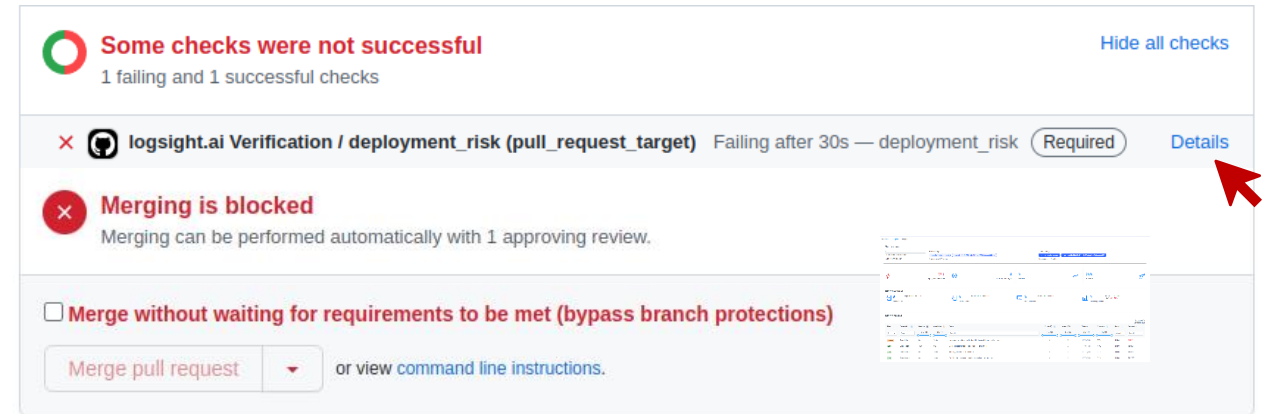


Back time
24h

ERROR: Cannot connect ...	\$
WARN: Low memory ...	\$
INFO: Reading file ...	\$
INFO: Socket created ...	\$



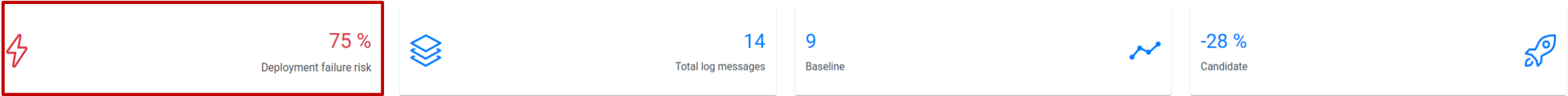
Developers push the code for verification



automated **verification** to suggest review
if the code contains **bugs**.

Developers see all relevant details about detected bugs

Risk of crash if deployed?



State overview



State analysis

Baseline (B) Candidate (C)										
Risk ↑↓	Description ↑↓	Baseline ↑↓	Candidate ↑↓	State ↑↓	Count (B) ↑↓	Count (C) ↑↓	Change ↑↓	Coverage ↑↓	Level ↑↓	Semantics ↑↓
S... ▾	Search	0 to 100	0 to 100	Search	0 to 100	0 to 100	0 to 100	0 to 100	Search	Search
MEDIUM	Added state	0%	100%	Uncaught exception: invalid literal for int() with base <:NUM>: '\n'	0	1	100% (+1)	7.1%	INFO	FAULT
LOW	Deleted state	100%	0%	User successfully added to database: john_miller	1	0	-100% (-1)	7.1%	INFO	REPORT
LOW	Added state	0%	100%	Starting calculation of the costs	0	1	100% (+1)	7.1%	INFO	REPORT
LOW	Added state	0%	100%	Starting the process of calculating total cost for a month	0	1	100% (+1)	7.1%	INFO	REPORT

Most probable reason for the bug?

Redirect to Source Code



```
1.def calculate_total_cost():
2.     total_cost = 0
3.     logging.info(f"Starting the process of calculating total cost for a month")
4.     logging.info(f"Connecting to the database.")
5.     try:
6.         db = database.connect()
7.         logging.info(f"Connection to database is successful")
8.     except Exception as e:
9.         logging.error(f"Database connection failed. Reason: {e}")
10.    logging.info("Starting calculation of the costs")
11.    while True:
12.        entry = db.read_entry()
13.        if entry.is_numeric():
14.            total_cost += int(entry)
15.        if not entry:
16.            break
17.    logging.info(f"Total cost of operation is: {total_cost}")
```

*Developers directed to the code containing the **bug** to quickly fix it.*

Update is ready to be deployed in production

Developers push the new code for **verification**.



All checks have passed

2 successful checks

[Show all checks](#)

Where are we today?

- Routine tasks easy to automate, but far away of challenging IT automation
 - Mathematical anomaly $\neq$ IT anomaly
 - Balancing sensitive and helpful detection $\Rightarrow$ 10.000 false alerts per day not useful
 - Explainable detection and remediation
 - How to scale to 10-1000x more devices in IoT and edge scenarios?
 - Transferable systems managing cold-starts and upgrades?
 - Acceptance by human-operators and users?
 - Cost of maintain and upgrading AIOPs platforms?